DEEP-SUBMICRON EMBEDDED PROCESSOR ARCHITECTURES FOR HIGH-PERFORMANCE, LOW-COST AND LOW-POWER

THÈSE N^{O} 3742 (2007)

PRÉSENTÉE LE 26 AVRIL 2007

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR

Laboratoire de traitement des signaux

SECTION DE GÉNIE ÉLECTRIQUE ET ÉLECTRONIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Sylvain AGUIRRE

ingénieur diplômé de l'Institut des Sciences de l'Ingénieur de Montpellier, France
et de nationalité française

acceptée sur proposition du jury:

Prof. Y. Leblebici, président du jury
Prof. D. Mlynek, Dr M. Mattavelli, directeurs de thèse
Prof. B. Hochet, rapporteur
Prof. M. Kunt, rapporteur
Dr D. Nicoulaz, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Lausanne, EPFL
2007

Abstract

Because the market has an insatiable appetite for new functionality, performance is becoming an increasingly important factor. The telecommunication and network domains are especially touched by this phenomenon but they are not the only ones. For instance, the automotive applications are also affected by the passion around the electronic devices that are programmable.

This thesis work will focus on embedded applications based on programmable processing unit. Indeed, nowadays, for time to market reasons, re-use and flexibility are important parameters. Consequently, embedded processors seem to be the solution because the behavior of the circuit can be modified by software what does not cost a lot compared to Application Specific Integrated Circuits (ASICs) or Digital Signal Processors (DSPs) where hardware modifications are necessary. This choice is judicious compared to multi-pipeline processors like superscalar or Very Long Instruction Word (VLIW) architectures or even in comparison to a Field Programmable Gate Array (FPGA) which require more silicon area, consume more energy and are not as robust as simple scalar processors.

Nevertheless, commercial scalar processors, dedicated to embedded systems, have poor frequencies which has a negative effect on their performance. This phenomenon is even more visible with deep-submicron technologies where the primary memories and wire delays do not scale as fast as the logic. Furthermore, the memory speed decreases when their capacity of storage increases and depends on both their organization (associativity, word size, etc.) and the IPs of the foundry. Likewise, synthesizable IP memories have a greater access time than their hard macrocell counterparts.

This thesis work proposes a new synthesizable architecture for scalar embedded processors dedicated to alleviate the drawbacks previously mentioned and called *Memory*

Wall: so, its goal is to push back the limits of frequency without introducing wasted cycles used to solve data and control dependencies, independently of the foundry. The architecture that came out, called *Deep Submicron RISC* (DSR), is made up of a single pipeline with eight stages that executes the instructions in-order. In addition to tackle the memory access time and to alleviate the delays of wires, it is appropriate to minimize the power consumption.

The proposed architecture is compared to two MIPS architectures, the MIPS R3000 and the MIPS32 24k in order to be able to analyze the performance of the architectures themselves, independently of both *Instruction Set* (ISA) MIPS 1 and compiler. The R3000 is a processor born in the 90's and the 24k came out in 2004.
Obviously, the study reveals that the five-stage processor remains efficient—especially in comparison to the MIPS24k—when the critical path passes by the core and not by the primary memories.
Even if the MIPS24k tackles in part the *Memory Wall*, DSR is much more efficient and reach a gain of efficiency—defined as $\frac{performance}{surface}$—of 72% thanks to its High-density version (DSR-HD) compared to a five-stage processor. DSR is even more efficient than the two MIPS processors when the transistor channel length decreases, the wire delays are important or the memories are large and their organization complex.

Key words:
embedded processor, scalar architecture, memory wall, high-efficiency, low cost, deep-submicron technology, interconnexion delay effect.

Résumé

Devant la démocratisation de l'accès aux technologies de l'information, les besoins en capacité de traitement des données, en vitesse de traitement et de transport de la dîte information ainsi que l'exploitation des services associés ne cessent de croître. Les domaines des télécommunications et des réseaux sont particulièrement concernés par ces phénomènes, mais ils ne sont pas les seuls. A titre d'exemple, le secteur automobile est également touché par l'engoûement que suscite l'électronique associée à l'informatique.
Ce travail de thèse se concentrera sur les applications embarquées basées, de surcroît, sur une unité de traitement programmable. En effet, pour des raisons de mises sur le marché rapides et afin d'éviter de re-développer une majeure partie d'un circuit, la notion de flexibilité est cruciale de nos jours. Par conséquent, les processeurs embarqués semblent tous désignés pour répondre à ces attentes, moyennant des modifications logicielles rapides et peu coûteuses comparées à des *Circuits Intégrés Dédiés à une Application* (ASIC) ou à des *Processeurs de Traitement du Signal* (DSP) où des changements matériels s'imposent. C'est aussi un choix judicieux en comparaison à des processeurs à plusieurs chaines de traitement d'information (*pipelines*) comme les architectures à *Longues Instructions* (VLIW) ou *superscalaires* ou encore, à des *Matrices de Portes Logiques Programmables* (FPGA) car ils impliquent peu de silicium, ils consomment peu et ils sont fiables de par leur simplicité.

Néanmoins, force est de constater que les processeurs du marché, destinés à des systèmes embarqués, ont des vitesses de fonctionnement très limitées, ce qui a un effet néfaste sur leurs performances. Ce phénomène sera d'autant plus notable avec le passage aux technologies dites *à longueur de canal inférieure au micron* ou *deep submicronique* pour lesquelles l'évolution des vitesses de fonctionnement des mémoires primaires et des interconnexions ne suivront pas celles de la logique intégrés. Qui plus

est, la fréquence de ces mémoires décroît quand leur capacité de stockage augmente, augmente en fonction de la complexité de leur organisation (associativité) et dépend des librairies du fondeur. Leur temps d'accès est également plus important lorsqu'il s'agit de *Solutions Logicielles Propriétaires* (IPs) synthetisables en comparaison à des circuits déjà intégrés dans une technologie donnée (*hard macrocell*).

Ce travail de thèse propose une nouvelle architecture synthétisable pour processeurs scalaires embarqués destinée à éviter ou atténuer, suivant les cas, les inconvénients précités, regroupés sous le terme de *Mur de Mémoire* (*Memory Wall*) : elle a donc pour objectif de repousser les limites de vitesse de fonctionnement tout en limitant l'impact de dépendance des données et du contrôle du flot d'execution d'un programme sur la performance, quelle que soit la fonderie. L'architecture résultante, nommée *Deep Submicron RISC* (DSR), est un unique *pipeline* à huit étages qui traite les instructions dans l'ordre dans lequel elles arrivent. Outre le fait de s'attaquer aux temps d'accès des mémoires *on-chip* ainsi qu'aux délais de propagation des signaux dans les interconnexions, on pourra démontrer, a posteriori, qu'elle a également un effet positif sur la consommation d'énergie.

L'architecture proposée DSR est comparée à deux architectures de la compagnie MIPS, le MIPS R3000 et le MIPS32 24k, afin de pouvoir comparer les performances liées à une architecture, indépendamment du compilateur et du *jeu d'instructions* (ISA). En effet, DSR et son implémentation ont été réalisés à partir du *jeu d'instructions* MIPS 1. Le R3000 étant un processeur standard à 5 étages de *pipeline* du début des années 90 et le 24k le dernier modèle de chez MIPS datant de 2004.
L'étude montre, s'il en était besoin, que lorsque le temps d'accès de la mémoire primaire ne pénalise pas la fréquence intrinsèque du processeur, l'architecture à 5 étages reste efficace, notamment vis-à-vis du MIPS32 24k.
Même si le 24k tient partiellement compte de la problématique étudiée dans ce travail de thèse, il n'en est pas moins que DSR atteint un gain en efficacité ($\frac{performance}{surface}$) de l'ordre de 72% grâce à sa version *Haute Densité* (DSR-HD) par rapport à son équivalent à 5 étages. La DSR se démarque d'autant plus des deux processeurs MIPS que la longueur de canal du transistor diminue, le délai des interconnections est important, la taille des mémoires est grande et leur organisation est complexe.

Mots-clés :
processeur embarqués, architecture scalaire, mur de mémoire, haute efficacité, faible coût, technologie à longueur de canal inférieure au micron, impact des délais des interconnexions.

Acknowledgements

Just a few words to express my gratitude to all the people, both near and far, who were involved during this thesis work period.

First, I would like to thank Professor Daniel Mlynek, my thesis adviser for giving me the opportunity to do my thesis on a hot topic in his laboratory, and to develop my skills in the industry world. His open mind and solutions he found during this period greatly helped to provide a good working environment.

I particularly would like to thank Daniel Auvergne, Michel Robert, Lionel Torres and all the staff at the LIRMM and Polytech in Montpellier who trusted and helped me in creating a partnership with the EPFL.

I next offer my thanks to people deeply involved in this project, Eduard Kohler and Yves Cheant of the EIVD at Yverdon for whom hardware implementation has no secret. I must thank Harsh Mehta and Vaneet Aggarwal from the Kanpur University in India who did a great job with the software. I wish you success on your own PhD work.

I am especially grateful to Marc Epalza for all the reviews and accurate comments he made on my work. Thank you for the time you dedicated to me on discussing weird questions. It was my pleasure to be your companion in the industry and then at the EPFL. I wish you all the best in your future position.

My warmest thanks go to my friends, my parents, my brother Florent, and above all my sons Hugo and Romain, and my wife Carole, for supporting me during this long period.

I want to acknowledge Christian Berg for answering my questions about the last MIPS's processor architecture, Paolo Ienne and Markus Levy for giving me access to the EEMBC's benchmark and its environment as well as for their support.

To Filip Caenepeel and Massimo Ravasi: thank you for your help and support, especially in generating tremendous numbers of SRAMs and settling test environments.

I also wish to thank Aline Gruaz, Alain Vachoux, Gilles Auric, Ali Djelassi, Marie Halm and Paul Debefve for their assistance in administrative tasks, in setting VLSI tools and keeping the PC network alive day after day.

Finally, I thank all the folks at the LTS3 laboratory for their varied help and companionship.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Superscalar or VLIW processors are usually not dedicated to embedded systems because of their cost (silicon area) and their energy consumption. Indeed, simpler and smaller machines like single instruction issue processors are more appropriate for this market [4]. Moreover, embedded applications that need high processing power run on a large number of microprocessors such as a matrix of simple *Central Processing Units* (CPUs).

Furthermore, one can note that all of today's IP embedded processors run at relatively low clock rates [30] and because not all embedded applications are concerned by power consumption, elaborating a higher performing embedded RISC processor architecture makes sense. One way of doing this is to increase the clock frequency of the processor—i.e., the number of operations performed per second. To do so, we had to understand how pipelined logic could be so slow: the bottleneck comes from the *on-chip* primary cache access time, also known as the *Memory Wall* [14] [29] [32]. Furthermore, this critical path will worsen with new technology nodes: indeed, the speed gap between logic and memory access will continue to increase for each new technology node because the wire delay will not scale down as much as the transistor delay [9][31]. Consequently, a new architecture should integrate this undesirable effect. Higher performing scalar processors will be interesting for embedded applications that need a high processing power like network processors that run on arrays of microprocessors: in this case, the matrix would contain less processors because each processor has a higher computation power. Consequently, less silicon area will be used and the

program partitioning among those CPUs would be easier to do.

Although that bottleneck was also newly highlighted by analysts [59], it did not seem to be really taken into account by scalar processor manufacturers as we will see in chapter 2.

Finally, this work is well suited for fabless companies that integrate CPUs in their chip. Indeed, in addition to the benefit of building the chip the way they want (package, speed independent) and being independent of the processor vendor's health, they should still work with the chip factory they like to work with. The latter point is essential in this study: indeed, there is great difference of frequency between on-chip memories depending on the foundry one chose. Consequently, if they do not want their chip to be limited by the memory access time, they could be forced to choose among a very limited number of chip factories which propose their own on-chip memories in the sense that they are already included in their processor cores: in that case, fabless companies may be constraint to buy the whole CPU and memories package.
The processor architecture developed in this document allows the fabless chip makers to license both CPUs and primary memories from different IP vendors with no constraint on the foundry choice and may ensure better performance, as detailed in chapters 4 and 7.

Furthermore, a side effect of this thesis work is to reduce the performance gap between embedded microprocessors designed by large teams and those created by small companies as explained in paragraph 7.2.

1.2 Goals

The goal of this work is to propose a High Performance and High Density RISC architectures, called DSR-HP and DSR-HD, that eliminate or alleviate the primary on-chip memory critical access time effect. To do so, comparisons will be made with processors based on the same ISA and using a single C compiler (with the same options).

This timing is increasingly critical with the new technology nodes because primary memories are still accessed in one clock cycle whereas long wires in the cache's critical path will prevent cache delay from scaling as quickly as simple gate delays: it is called *Deep Submicron RISC* architecture because DSR tackles the deep submicron wire penalty.

Two other different processor architectures will derive from DSR-HP based upon that unique microarchitecture. Each of which treats a specific integrated circuit factor like performance, silicon area or power consumption, as mentioned below:

1. DSR-HP, a *High Performance* (HP) architecture, which is the starting point of this thesis. Performance improvements, in comparison to MIPS's processors, due to an increase in the clock frequency until a certain speed threshold be reached. DSR can even use High Density (HD), and then slow, primary memories in certain applications and still ensure a performance gain. The two following architectures derive from it.

2. DSR-HD, a HD architecture based on high density memory technology but that uses single port primary memories instead of dual ports, this time. This architecture requires less silicon area than the previous one because dual-port memories are bigger than their single port counterparts. Useful when the performance is not a priority or when these single-port slow memories still allow to exceed a frequency threshold that increase performance. For efficiency reasons, the thesis will gradually focus on this architecture.

3. DSR-LP, a Low Power architecture similar to DSR-HD but where supply voltage scaling is applied to feed the memories. Indeed, since DSR needs two clock domains and allocates the slowest one to the primary memory subsystem, the caches' supply voltage can be lowered until a value that correspond to the required cache's frequency [19]. This architecture cumulates two advantages from a power consumption view point: first, the primary memory clock is slowest than the CPU for an unchanged number of access, then, the power supply can be lowest than the CPU.

Obviously, the choice of one of these architectures depends on the application needs. Thus, the main challenges of this work consist in improving the performance of the processor designed in submicron technologies independently of the foundry and allowing the system to use or keep using high-density, slow memory components—such as caches or SRAMs—to enhance the efficiency.

1.3 Structure of the Thesis

The first part of this thesis deals with a brief survey of current commercial and academic scalar embedded processors in its second chapter: most well-known processors which treat instructions in order are analyzed.
Chapter 3 discusses the limitations of such architectures in terms of speed and performance, leading to the proposed architecture described in chapter 4—architecture that is still scalar and the stages of whom are described in chapter 5.
Chapter 6 describes the software environment used to model, test and characterize our solution and to implement it. Indeed, a VLSI implementation of the proposed architecture was achieved and the circuit area and speed features were exploited to elaborate the results reported in chapter 7.
Performance results and comparison with its counterparts are reported in chapter 7 where memory size and organization, interconnexions and deep submicron technologies are taken into account.
Finally, chapter 8 concludes this report and gives perspectives for future jobs.

Chapter 2

Embedded Scalar Processor State of the Art

2.1 Introduction

The embedded microprocessor market is the fastest growing portion of the computer market [15][20] in the communications, data processing infrastructures, medical, instrumentation and factory automation segments. ARM [54] should still remain the most important embedded processor vendor, in terms of market shares, among others MIPS [62], ARC [53], IBM [58] and Tensilica [65] competitors for the near future.

Even recently, most embedded scalar processors were made up of five stages unified on a single pipeline for power and cost reasons [20]. So, ARC [53], ARM [16], MIPS [27], PowerPC [28] or other Tensilica [65] processor vendors dedicated a single clock cycle to access both instruction and data memories. As a consequence and claimed in [29], [9] and resumed by the Faraday company [14], the frequency of current scalar processors is limited by the technology of today's on-chip memories that have an access time that do not follow the speed evolution of their logic counterpart. Phenomenon that will worsen with the new technology nodes due to the wire delay that will become increasingly prevailing [1] [12] [22] [60].

Table 2.1 reports the main recent embedded processors on the market[1]. The trend is to extend the number of stages (traditionally set to five) but not for the same purpose: only some vendors allocate more time to access the memory. All these companies claimed to reach high frequency such as 550 MHz and we can believe them

[1] A quite exhaustive list of microprocessor vendors and cores is located at [61]

if they are speaking about the bare CPU because the logic is well balanced over all the stages. However, this is just a marketing technique because a core is useless without associated memories, and then these numbers have no real value. This is even truer when one knows that primary memory's data that go back to the core are part of the critical path because memory typically operates at slower speeds than processor clock rates.

The clock frequencies reported in table 2.1 were mainly obtained with the TSMC 0.13μm processes under the worst conditions: the *LV-OD* process, based on Low Threshold Voltage, is used for high performance purposes whereas *G* defines a standard and low-cost process. All these feature are provided by the companies themselves through their web site and concerns the CPU core only, i.e., without memories.

CPU cores	**CPU Frequency (MHz)**		**Pipeline Depth**	**Instruction Issue**
	CL013G	CL013LV-OD		
MIPS32 24K	400	625	8	1
MIPS32 4Kc	210	330$^{(a)}$	5	1
ARM11	333	550	8	1
XtensaLX	300	350	7/9$^{(b)}$	2
XtensaV	300	350	5	1
ARC700	400	533	7	1
ARC600	290	350	5	1
PPC405 GPr	266 – 400$^{(c)}$		5	1
PPC440 EP	333 – 533$^{(d)}$		7$^{(b)}$	2
PPC440 GP	400 – 500$^{(c)}$		7$^{(b)}$	2
PPC440 GX	533 – 800$^{(d)}$		7$^{(b)}$	2

Table 2.1: Main Embedded Processor Core (only) Features. (a) frequency obtained by an extrapolation of 56.25% from 210 MHz using the relation between the MIPS32 24K frequencies under the CL013G and CL013LV processes. That is the faster MIPS 5-stage processor. (b) superscalar architectures. (c) results obtained with a CMOS SA-27E, 0.18μm IBM's process (Hard macro). (d) results obtained with a CMOS Cu-11, 0.13μm IBM's process (Hard macro).

2.2 Academic's Solutions

Since a while, memory has been identified as a bottleneck for processor performance and the term of *Memory Wall* came out [51] which includes the memory bandwidth, the miss rate, the miss penalty and the hit time in the cache. Even if many researches have been made on these factors, the *Memory Wall* issue is still of topicality [32] and

the L1 memory access time remains a major limiting factor for embedded systems [40], especially for cacheless applications. What follows mainly focuses on what have been done, up to now, to reduce the L1 cache access time.

As of the 80's, the processor-memory gap was born in the sense that memory access time started to slow down the CPU execution time [20]. Then, *caches* was introduced to alleviate this effect [46]: this approach is based on the fact that smaller hardware is faster and on the *principle of locality*. The idea behind the latter point is that the required data are mainly located in a close range of addresses within the memory. Consequently, a small memory should contain the requested data and whenever it is not the case, the main memory is solicited.

Then, still for performance and cost reasons, appeared the principle of *memory hierarchy* that consists in having multiple levels of caches between the CPU and the main memory, where lower is the level, smaller and faster is the cache [46]. This technique may help to reduce the memory hit time by having even smaller L1 caches.

In the 90's, having small caches allowed to implement L1 caches on the same chip that the processor and then reduced the cache access time [48].

Also, it came out that simple caches have the smallest access time due to the fact that the data may come from a less important number of lines—e.g., the data in a one-way set associative, also called direct-mapped, cache is located in a single line.

The previously mentioned multi level memory system may also have a negative effect on the L1 data cache access time. Indeed, whenever a memory request occurs, the CPU checks if this request can be satisfied by the L1 cache (excepted for cacheless applications) by comparing a part of the current address—referred as tag—with the ones of the L1 cache. Generally, the program address needs to be translated in a physical address to access the memory. Depending on the program size and the cache organization, this translation may take time and slow down the processor clock rate by increasing the cache access time. So, a memory access is divided into two parts: an access to a small cache—called *Translation Lookaside Buffer* (TLB) [3]—that stores tags of addresses present in the L1 data cache to determine if the required data is in the L1 data cache (called *hit*), and second, an access to the L1 cache itself.

The *decoupled cache* is a technique established to avoid or alleviate an extra L1 cache access time penalty due to memory address translation and cache hit detection:

it consists in having a fast associative cache access by allowing to fetch the requested data from the cache without waiting until all the tag comparisons are achieved—knowing that there is no tag comparisons for direct-mapped memories since a cache line has only one place in the cache. To do so, the data cache and the TLB are accessed in parallel, and the fetched data will be discarded in the pipeline if the tag was not in the TLB (miss). Generally, an one to eight-entry FIFO is used to store write instructions because in that case the hit detection cannot be done in parallel with the memory access: indeed, one has to be sure that the right data will be overwritten [20].

The end of this paragraph deals with processor architecture proposals of the last years that tackles the overall memory access time.

In 1996, processors tightly integrated into Dynamic Random Access Memories (DRAM) was proposed that allows to reduce memory access latencies since the main memory and the processor are on the same die [43].

In 2000, a processor architecture for high performance computing that intend to alleviate the performance gap between memory and processor was presented. Software Controlled Integrated Memory Architecture, SCIMA, uses an integrated on-chip memory in addition to the L1 cache of which the location and the replacement of the data are explicitly controlled by software instead of hardware implicitly. This on-chip memory stores future required data that will not be flush out from the cache in case of location conflict which leads to performance degradation [37].

The same year, an architecture with an on-chip SRAM, in addition to the L1 data cache, was proposed to store scalar and arrayed program variables and thus, avoid to fetch them from a high-level off-chip memory [38]. This proposal allows to reduce the miss cache latency.

Even if what follows is not a pure academic research, it was published in ISSCC'02 where Matsushita Electric Industrial disclosed its 32b embedded microprocessor core [36]. It is a single instruction issue, in-order RISC processor with a variable instruction length claimed to be made up of seven stages—even if it better looks like an eight-stage architecture—where cache accesses take two stages (MEM 1 and MEM2) as illustrated in figure 2.1. However, only MEM1 is dedicated to fetch the information

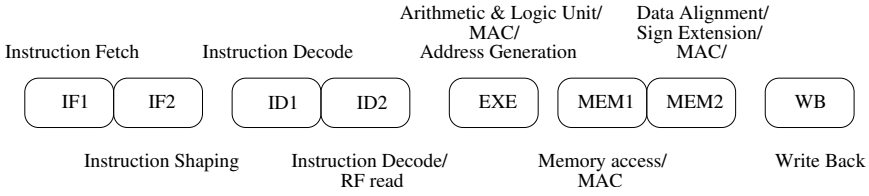

Figure 2.1: AM34-1 Microarchitecture Block Diagram: the pipeline performs at most one instruction per clock cycle and treats instructions in-order.

from the data cache whereas MEM2 shapes the data.

In 2004, a Flexible Sequential and Ramdon Access Memory (FSRAM) has been elaborated to improve the main memory access time of off-chip memories. The principle mainly resides in bypassing the row address decoder during sequential accesses [10].

2.3 ARC Processors

ARC^{TM}'s configurable 32-bit RISC processors are mainly based on a five-stage core through their ARC 600 family, and their last ARC 700 family is a seven-stage scalar pipeline.
Whereas the ARC 600 peaks at 290 MHz in a TSMC CL013G process, the synthesizable ARC 700 can reach 400 MHz but requires nearly four times more silicon area [17]. This latter processor is made up of a dynamic branch predictor, a faster, single-cycle adder, wider memory interfaces, nonblocking load/store pipeline capability and out-of-order completion.
The ARC 700 microarchitecture (see figure 2.2) divides its decode stage in an instruction-align stage (due to a mix of 16- and 32-bit instructions) followed by a bare decode stage: thus, less time is wasted to detect fetched instruction boundaries and align them before the decoding step. A new select stage is used to determine which unit will give its value to the writeback stage and allows a full clock cycle to access the data memory: i.e., the data formatting is done in the writeback stage, implying two cycles to have access to the loaded operand. Furthermore, jumps and branches have a single-instruction delay slot.

The use of caches as memories is optional and their interfaces can be 64 bits wide. DSP extensions are possible and up to 54 general purpose registers may be used.

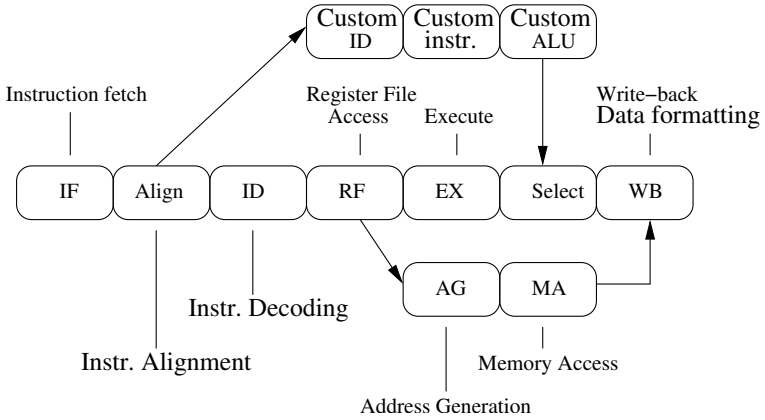

Figure 2.2: ARC Microarchitecture Block Diagram: it is a seven-stage scalar pipeline.

2.4 MIPS Processors

2.4.1 Five-stage Pipeline Architecture Overview

The MIPS® (Microprocessor without Interlock Pipeline Stages) company licenses RISC IP to semiconductor companies, ASIC developers, and system OEMs. Core families include the MIPS32 4K, 4KE, M4K, 4KSd synthesizable processor cores which are all 5-stage pipeline architectures dedicated to digital consumer, networking or security applications.

They are descendants of the MIPS R3000 [27] developed in the end of 80's. Figure 2.3 describes its microarchitecture, where:

- IF stage consists in fetching an instruction per cycle from the instruction memory,
- instruction that is then decoded in the ID stage, registers may be read and branches are solved,
- the ALU stage executes instructions. A data memory address can be computed in that step for memory instructions.
- Memory instructions access the data memory at the MEM step, otherwise the ALU result is just latched.
- Finally, the register file is updated in the WB stage, if needed.

Such an architecture (without branch predictor) suffers of 1-clock delay for both *load* and *branch/jump* instructions (we omit co-processor, multiplication/division operations). The cycle following a *load* and *branch/jump* instruction is called a *delay slot* and it is managed by the compiler which introduces a No-OPeration (*NOP*) instruction or an independent instruction in this slot. Indeed, the instruction following

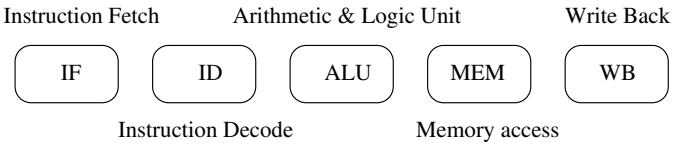

Figure 2.3: MIPS R3000 Microarchitecture Block Diagram: it is a standard five-stage pipeline made up of the elementary operations.

a *load* cannot use the loaded data from that *load*, even with forwarding path from the MEM stage back to the ALU stage as shown in figure 2.4.

Likewise, the instruction immediately following a *branch* is always executed, inde-

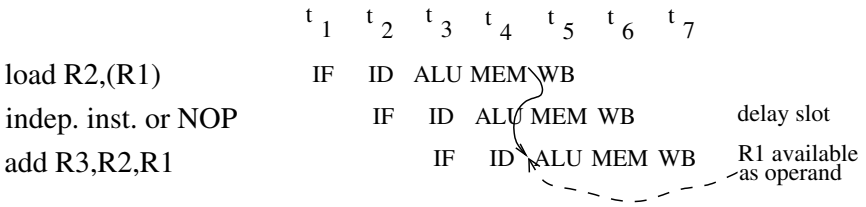

Figure 2.4: MIPS R3000's Data Dependency Resulting from a Load Instruction: the loaded R1 register can feed the ALU stage only at t_5. The instruction after the *load* instruction is in the *load delay slot* and do not read R1.

pendent of whether the branch is taken or not. The *branch delay slot* (t_2) is used to compute the branch destination address and fetch the target instruction (t_3) from the single-cycle instruction memory as shown in figure 2.5.

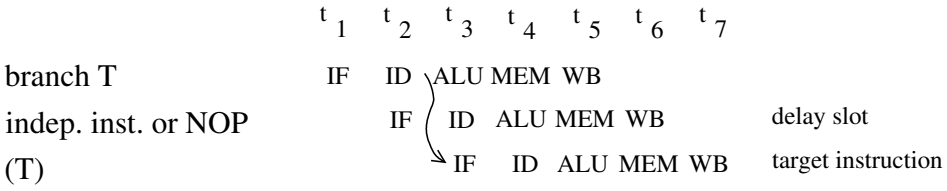

Figure 2.5: MIPS R3000's Data Dependency Resulting from a Branch Instruction: the destination address, *T*, is computed and provided to the instruction memory in the ID stage (at t_2) and the target instruction, *(T)*, is fetched at t_3. The instruction after the *branch* instruction is in the *branch delay slot* and it is always executed.

The CPU critical path starts at the 32-bit ALU adder, the forwarding multiplexer, the registers comparison for branch decision to end up at the program counter multi-

plexer.

2.4.2 Eight-stage Pipeline Architecture Overview

Now, we will focus on the latest architecture of MIPS that came out at the end of 2003, the MIPS32 24K. As shown in figure 2.6, it is a single-issue eight-stage pipeline[2] that implements a decoupled fetch unit used to perform dynamic branch prediction (claimed to be 90% accurate). The instruction cache (16, 32 or 64KB) transfers two 32-bit instructions (a bundle) per access through a 64-bit read port and feeds a six-entry instruction buffer used for the prediction. Likewise, the data primary cache is 4-way set associative and uses a 64-bit read/write port to communicate with the CPU. The core contains thirty-two 32-bit general purpose registers used for scalar integer operations and address calculation. The register file consists of two read ports and one write port and can be duplicated twice or three times for to minimize context switching overhead. To reduce the critical path due to L1 cache (compiled RAM) access, MIPS splits each memory stages IF and MEM, into two stages: then, a full clock cycle (DC stage) is allocated to access the data memory instead of 50-75% of a single clock cycle for 5-stage processor like the MIPS R3000.

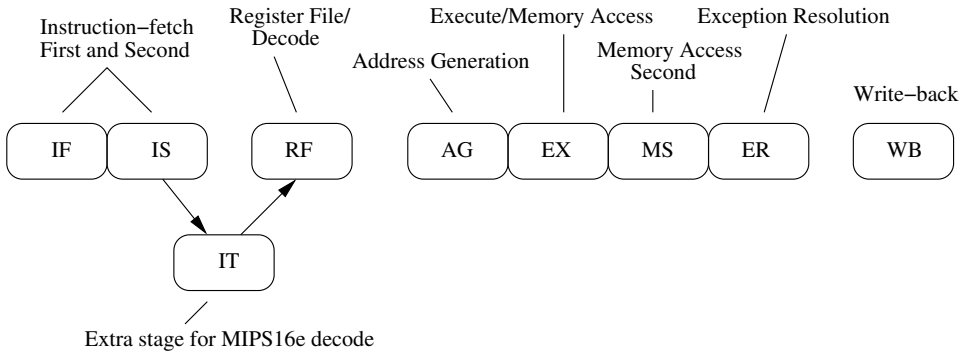

Figure 2.6: MIPS32 24K Microarchitecture Block Diagram: it is an eight-stage pipeline with an extra stage when processing MIPS16e instructions (That figure is extracted from [30]).

The description of each of MIPS24K's stages is as follows [33]:

- IF stage: Instruction Fetch First
  - I-cache tag/data arrays accessed
  - Branch History Table accessed

[2] A ninth stage is activated in case of use of the MIPS16 ISA to reduce code density will be ignored in the rest of this document.

  - ITLB address translation performed
  - EJTAG break/watch compares done

- IS stage: Instruction Fetch Second
  - Detect I-cache hit
  - Way select
  - MIPS32 Branch prediction

- IT stage: Instruction Fetch Third
  - Stage is bypassed when executing MIPS32 code and the instruction buffer is empty
  - Instruction Buffer
  - Branch target calculation

- RF stage: Register File Access
  - Register File access
  - Instruction decoding/dispatch logic
  - Bypass muxes

- AG stage: Address Generation
  - D-cache Address Generation
  - bypass muxes

- EX stage: Execute/Memory Access
  - skewed ALU
  - DTLB
  - Start DCache access
  - Branch Resolution

- MS stage: Memory Access Second
  - Complete DCache access
  - DCache hit detection
  - Way select mux
  - Load align

- ER stage: Exception Resolution
  - Instruction completion
  - Register file write setup
  - Exception processing

- WB stage: Writeback
  - Register file writeback occurs on rising edge of this cycle

The fetch unit operates autonomously from the rest of the machine, decoupled by an eight-entry instruction buffer. The processor reads two instructions from the I-cache (instruction cache) each cycle, allowing fetches to proceed ahead of the execution pipeline. Speculation accuracy is improved with a branch history table, holding 512 bimodal entries, and a four-entry returnprediction stack. A full cycle is allocated for the I-cache RAM access, with the cache hit/miss determination and way selection occurring in the following stage. One cycle is allocated to read operands from the register file and collect other bypass sources. Separate execution pipelines handle integer and load/store instructions. For memory operations, address calculation occurs in the AG stage. Next, the processor reads the data cache.The MS stage performs hit calculation, way selection, and load alignment. The processor can accommodate up to four non-blocking load misses, allowing hits under misses. Normal ALU operations pass through the AG stage and do their real work in the EX stage. This skewed ALU preserves the two-clock load-to-use relationship common to many other MIPS cores. The exception-recovery stage prioritizes any exceptions. Finally, the write-back stage updates the register file and other instruction destinations with new results [50].

Although most instructions can be issued at a rate of one clock cycle, such an architecture introduces extra wasted cycles in comparison to a shorter pipeline due to data and control dependencies that are even more prevalent. Typically, this phenomenon concerns load, branch and jump instructions as detailed below. Special instruction delays, like co-processor instructions, that are specific to a given architecture are beyond the scope of this study.

The MIPS32 24K architecture has the same notion of *load* and *branch delay slots* as previous MIPS five-stage processors and they are set to one clock cycle delay (see subsection 2.4.1).

2.4.2.1 Branch Latency

As a MIPS five-stage pipeline, the MIPS32 24K is followed by a delay slot that can be filled with a useful instruction when possible one otherwise a NOP is inserted. This is mainly due to an eight-entry instruction buffer and a *bypassable* IT stage that avoids extra cycles when a branch occurs. As illustrated in figure 2.7, two cases may happen: the delay slot is fetched in the same bundle as its corresponding branch instruction (figure (b)) or belongs to the next bundle (figure (a)). In both cases, instructions

are removed from the instruction buffer and a throughput of 1 instruction per cycle is ensured. The target instruction bypasses the IT stage because the buffer was empty.

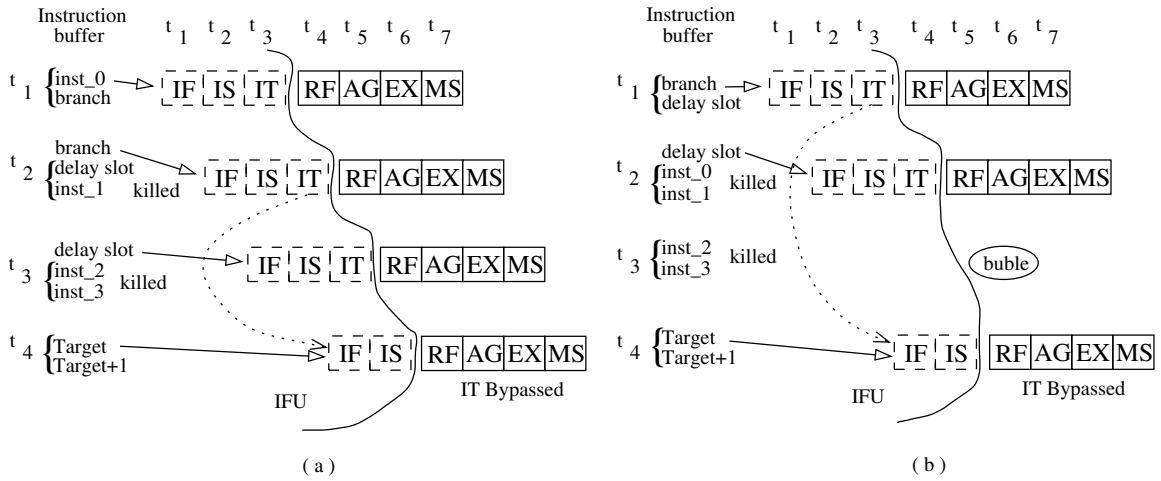

Figure 2.7: MIPS32 24K's Branch Taken Delay: (a) Branch Taken Delay when a branch instruction and its delay slot are not in the bundle. (b) Branch Taken Delay when a branch instruction and its delay slot are in the bundle. The Instruction Fetch Unit (IFU) issues instructions to the rest of the pipeline through it eight-instruction buffer.

In the case of a mispredicted branch being taken, there is a penalty of 4 clock cycles before the next valid instruction be executed (see figure 2.8).

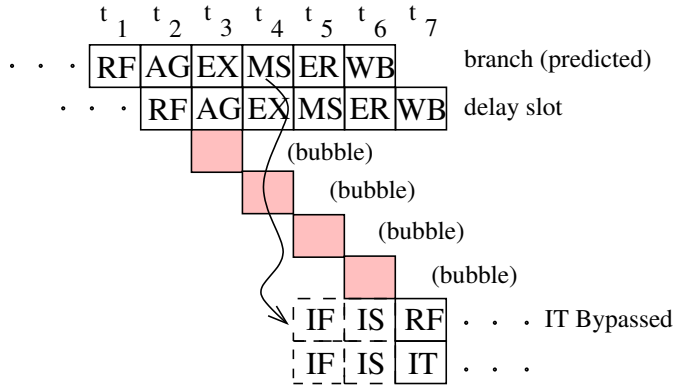

Figure 2.8: MIPS32 24K's Branch Taken Misprediction Penalty: The branch instruction is performed in the RF stage (at t_1), its resolution is done in the EX stage (at t_3) and used in the MS stage (at t_4). As a consequence, the next valid instruction is fetched at t_5 and performed in the RF stage at t_7 instead of t_3, t_2 being used by the delay slot: 4 clock cycles are wasted (bubbles).

2.4.2.2 Jump Latency

Jump instructions that specify a target address contained in a general register have a delay of 4 clock cycles in addition to the delay slot. This is due to the fact that the EX stage reads the register file. Then the target address is only available at the MS stage as is the case for a mispredicted branch (figure 2.8).

2.4.2.3 Load Latency

Two sequences can stall the pipeline:

- A load-instruction pattern as shown in figure 2.9 where one cycle is required by the dependent instruction to receive the data from the memory: this corresponds to the known *load* delay slot that we find in previous MIPS processors.

- A load followed by a load or a store pattern as shown in figure 2.10 where two cycles are required by the second memory instruction to receive the data from the memory: one extra cycle is necessary, in addition to the *load* delay slot.

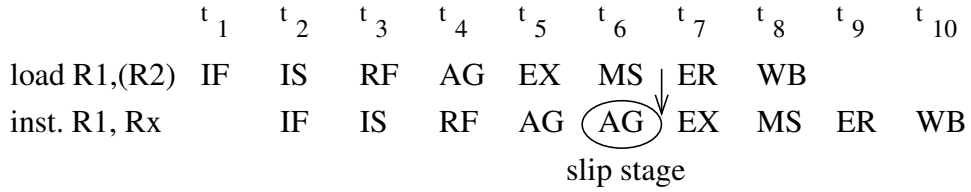

Figure 2.9: MIPS32 24K's load latency due to a next dependent instruction: the R1 register needed at the end of the MS stage (t_6) will be available only at t_7, introducing a one cycle delay.

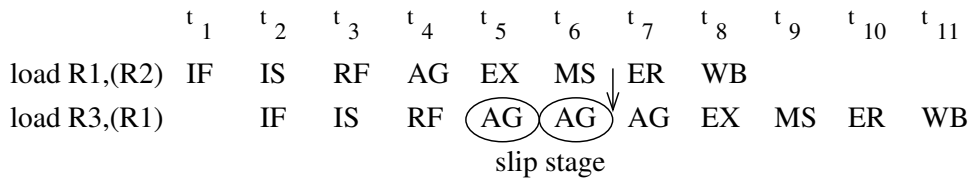

Figure 2.10: MIPS32 24K's load latency due to a next dependent memory instruction: the R1 register needed at t_5 will be available only at t_7, introducing a 2-cycle delay.

2.4.2.4 Miscellaneous Latency

When an instruction (different from a memory or a control instruction) updates a register that is read by a following memory instruction, the pipeline is stalled during one clock cycle because the AG stage of the *load* instruction waits after the result of the previous dependent instruction which is only available at the end of the EX stage as shown in figure 2.11.

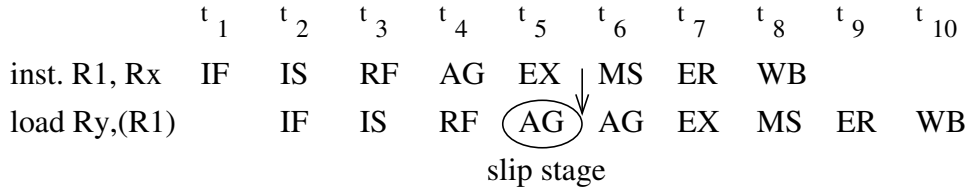

Figure 2.11: MIPS32 24K's latency for an instruction followed by a dependent memory instruction: an instruction, different of a memory or jump instruction, has a 1-cycle delay due to an immediate next dependent memory instruction: the R1 register needed by the *load* at t_5 will be available only at t_6, introducing a 1-cycle delay.

2.5 ARM Processors

ARM® (Advanced RISC Machines) is one of the most popular 32-bit embedded processor vendors of the market with their ARM7, ARM9 and ARM10 families, especially in the low-power applications like mobile phones .
ARM11 is the latest architecture family that came out of the ARM company by end of 2003. The goal was to achieve higher clock frequency than in the past to lead to a High-performance synthesizable microarchitecture. ARM claims to reach 550 MHz in a high performance TSMC process in 0.13μm (worst case - CL013LV) thanks to a deeper pipeline with their eight-stage scalar architecture (see figure 2.12).
Indeed, ARM faced the same bottleneck as MIPS—and all others scalar processors—that resides in the cache data path, returning data from the RAMs to the core. That is the reason why the memory access was split over two clock cycles. Despite the gain in frequency obtained, the memory access remains the critical path where 20% of the clock cycle is required to return the data to the CPU core: i.e., the minimum clock period is equal to the data cache access time plus $\frac{1}{5}$ of the minimum clock period [47]. The previous mentioned rule is true for 16KB or larger cache memories with a TSMC's 0.13G process because smaller memories are fast enough and the critical path is then

in the CPU itself. Consequently, like MIPS, ARM balanced the logic in all stages to fit the clock cycle imposed by the memory access stage, *DC1* [29]. Moreover, both 64-bit instruction and data caches are 4-way associative and can be set up to 64KB in size. The ARM11 core contains 32-bit registers.
An instruction buffer stands within the fetch unit to improve the timing of compiled RAMs and to predict branches with 85% of accuracy.

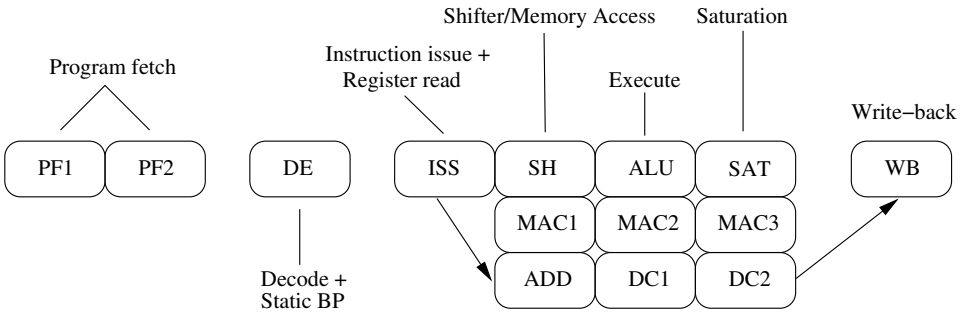

Figure 2.12: ARM11 Microarchitecture Block Diagram: that is an eight-stage pipeline (from [29]).

The description of each of ARM11's stages is as follows [6]:

- PF1: First stage of instruction fetch and branch prediction.
- PF2: Second stage of instruction fetch and branch prediction.
- DE: Instruction decode.
- ISS: Register read and instruction issue.
- SH: Shifter stage.
- ALU: Main integer operation calculation.
- SAT: Pipeline stage to enable saturation of integer results.
- MAC1: First stage of the multiply-accumulate pipeline.
- MAC2: Second stage of the multiply-accumulate pipeline.
- MAC3: Third stage of the multiply-accumulate pipeline.
- ADD: Address generation stage.
- DC1: First stage of Data Cache access.
- DC2: Second stage of Data Cache access.

- WB: Write back of data either from the Load Store Unit or from the multiply or main execution pipelines.

The ARM11 has separate execute and load/store pipelines that may run in parallel and thus lead to *out-of-order* completion.

2.5.1 Branch Latency

The ARM11 uses dynamic and static branch predictors. The dynamic branch predictor is based on a historical record of previous branch status thanks to a 64-entry branch target address cache (BTAC). In case the branch is not encountered in that local memory, the hand is given to the static branch predictor which follows a *backward taken* rule. Figure 2.13 shows a misprediction penalty.

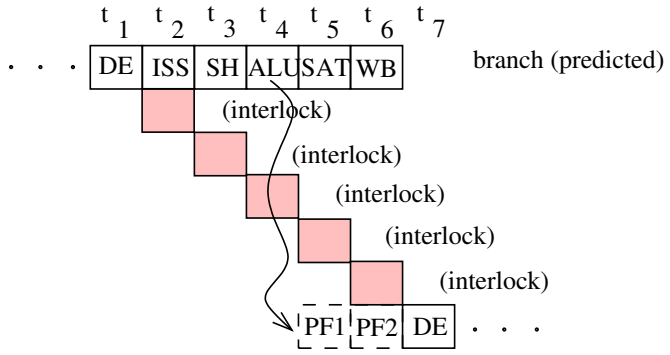

Figure 2.13: ARM's Branch Taken Misprediction Penalty: the branch is resolved at the ALU stage (t_4) and the next valid instruction will enter the DE stage at t_7 instead of t_2, implying a penalty of five cycles. Unlike MIPS with their *delay slot*, ARM uses hardware to delay an instruction until something is ready: the pipeline is then interlocked.

2.5.2 Load Latency

This section illustrates a few data dependencies induced by load instructions.

The use of pointers can lead to the sequence shown in figure 2.16 that implies a delay of two clock cycles.

2.5.3 Miscellaneous Latency

The complexity of the ARM11 architecture generates many dependencies—even with instructions other than memory or control instructions—and induces many cycle delays like those shown in figure 2.17 and 2.18.

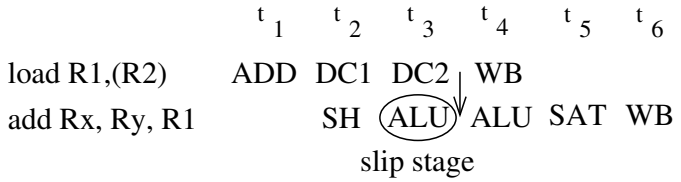

Figure 2.14: ARM's Load Latency due to a next dependent instruction (1/2): the second instruction needs its operands at t_3 (ALU stage) but the R1 register is only available at the end of the DC2 stage (t_4). Data forwarding to the ALU stage allows a single stall.

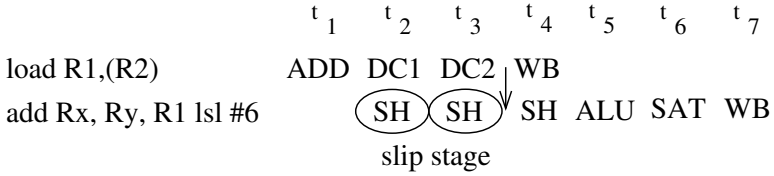

Figure 2.15: ARM's Load Latency due to a next dependent instruction (2/2): the second instruction needs its operands at t_2 (SH stage) but the R1 register is only available at the end of the DC2 stage (t_4). Then, the SH stage is slept during two clock cycles, which creates a penalty of two clock cycles.

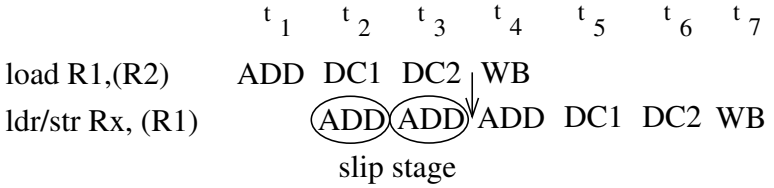

Figure 2.16: ARM's Load Latency due to a next dependent memory instruction: the second instruction needs its operands at t_2 (ADD stage) but the R1 register is only available at the end of the DC2 stage (t_4). Thus, the ADD stage is stalled for two clock cycles.

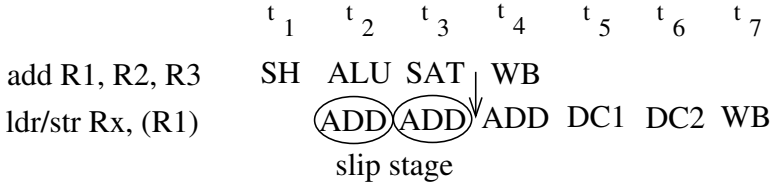

Figure 2.17: ARM's Latency for an instruction followed by a dependent memory instruction: the second instruction needs its operands at t_2 (ADD stage) but the R1 register is only available at the end of the SAT stage (t_4). Hence, the ADD stage is idled until (t_4) and generates a delay of two clock cycles.

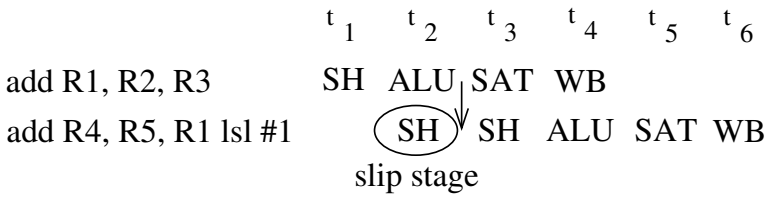

Figure 2.18: ARM's Latency between two dependent instructions: the second instruction needs its operands at t_2 (SH stage) but the R1 register is only available at the end of the ALU stage (t_3). One stall cycle is required.

2.6 XTENSA Processors

Tensilica® addresses the market of configurable embedded processors with its Xtensa V and Xtensa LX synthesizable cores.

The Xtensa V is a scalar five-stage architecture that mixes 16-, 24- and 32-bit instructions. Consequently, a loaded register is available two cycles after the current load instruction. Furthermore, two cycles are wasted for conditional taken branches due to the fact that the condition is checked at the EXE stage: a third cycle is required whenever the target instruction to execute does not fit within a 32-bit word of the instruction memory.

An interlock mechanism is used to stall the pipeline and a windowed register file is implemented to accelerate function calls.

The last architecture designed by the Tensilica company is the Xtensa LX, a configurable VLIW processor made up of seven stages that can issue 2 instructions per cycle (see figure 2.19). The pipeline is configurable in the sense that the number of stages may change. Indeed, by default, the Xtensa LX pipeline has five stages, like the Xtensa V, that may be extended to seven, adding an extra instruction-fetch stage and extra data-access stage. The resulting architecture is planned to tackle slow on-chip memory access time by giving twice (or even three times) more time to access the primary memories [18]: more precisely, the access time should be about 1.5 and 1.8 clock cycle due to wires and multiplexers that return the data back to the core. The pipeline synchronization between different clock domain is achieved through an interlock mechanism that stalls the pipeline. Because the processor core is automatically generated by Tensilica's tools, the logic is not balanced over the numerous stages as one can expect for a deeper pipeline: as a consequence, the Xtensa LX does not run faster than a five-stage pipeline.

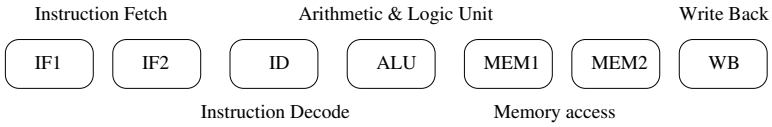

Figure 2.19: Xtensa LX Microarchitecture Block Diagram: it is a seven-stage VLIW pipeline which can deliver up to two instructions per cycle.

2.7 PowerPC Processors

Two main families cover the PowerPC®'s hard macro embedded processors: the 405xx and 440xx cores. Whereas, the 405 family is a five-stage scalar architecture, the last 440 is a seven-stage dual-issue superscalar architecture.
The 405 core is a five-stage with all the elementary operations such as *Instruction Fetch*, *Instruction Decode*, *execution*, *Memory Access* and *Write-back* and where the data cache is still accessed in one clock cycle like in the new 440 family. Furthermore, its 32-bit general purpose register file has three read ports and two write ports to be able to execute load or store instruction in parallel with an ALU operation. The fetch unit uses a static branch predictor where backward branches are chosen as taken: in case of a missprediction, a maximum of 5 clock cycles are wasted.
Compared to it, the superscalar 440 core (figure 2.20) uses a dynamic branch predictor and sometimes requires extra clock cycles for instructions that read the register loaded by a preceding load instruction.

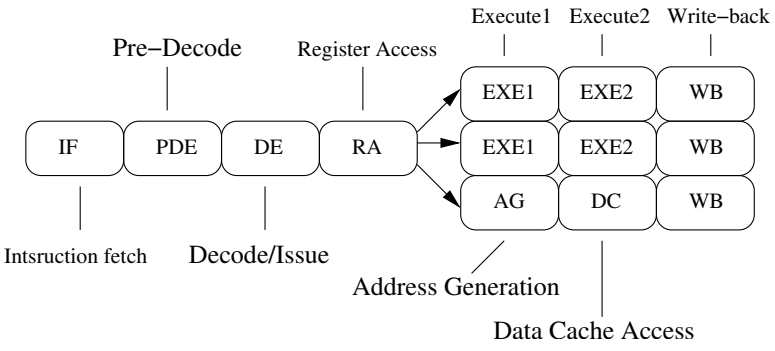

Figure 2.20: PPC440 Microarchitecture Block Diagram: it is a seven-stage superscalar out-of-order pipeline which can deliver up to two instructions per cycle and uses dynamic branch prediction.

We can note that all cores are delivered with both their instruction and data L1 caches (16KB or 32KB each).

2.8 Conclusion

Many researches were achieved at the memory level that reduce the access time of the CPU closest memory but no solution includes the processor architecture as a parameter (excepted the AM34-1 processor that has the same approach than the ARC700, the MIPS 24K and the ARM11).

We will not mention the PPC440 processor any further because, in addition to be an energy-hungry core, it is a superscalar architecture that did not make any improvement in the main bottleneck which is the data memory access time.
Even if Tensilica, with its Xtensa LX, has had a good approach to tackle this issue, the exploitation of their VLIW architecture is far to be optimal because we face to a pipeline that introduces wasted cycles due to its extra pipeline depth and that runs at the same slow frequency as a five-stage processor.
The ARC 700 is close to the MIPS32 24K from the data cache memory access point of view in the sense that a full clock cycle is given to achieve this task.
MIPS and ARM, which are the most dynamic companies in the scalar processor domain, propose their new MIPS 24K and ARM11 architectures that bring a partial solution to the memory access time: between half and a quarter of cycle is added to the former 5-stage memory access time that represents an increase of 25-50%. So, only between 75 and 100% of the clock cycle is allocated to return data from the primary data memory.

However, these architectures can ensure enough time both to propagate the address out to the ram before the access for processors that would like to run to high frequency such as 550 MHz (in a TSMC 0.13μm high performance process, CL013LV). Nevertheless, the processor frequency is still limited by the cache frequency which varies with its size, its structure, the semiconductor process and the libraries for the RAMs (RAM compilers) used in the cache implementation. Indeed, MIPS admits that 64KB caches slow down the MIPS24K frequency in most processes [34] and the ARM11's frequency saturates from a 16KB cache size configuration [47]. Moreover, it just reaches 333 MHz with 16 KB Virage Logic's memories where the core was expected to overcome 400 MHz on the TSMC's 0.13G process [13]. Consequently, fabless companies who get processor and on-chip memory licenses to achieve System-On-Chips (SoCs) on arbitrary foundries are still limited by the memory technology that does not follow the CPU speed. In these conditions, the use of high density components is even more

compromised.

Chapter 3

Embedded Processor Limitation

As described in the previous chapter, all the scalar architectures do not allocate more than a single clock cycle to access the primary memories. Primary on-chip memories of today's embedded processors limit the performance of these systems for three main reasons:

- all required data are not (always) located in single-cycle caches or SRAMs, implying data cache miss latency,
- larger caches can be used to reduce the previous mentioned effect but large caches are slow,
- and only between half and two thirds of a CPU clock cycle is left to access the data cache memory (t_{cq}) in a 0.13μm due to wire delays and forwarding multiplexers.

Furthermore, the delays induced by the wires will be increasingly important with the future technology nodes (90nm, 65nm, 45nm, 32nm, 22nm, ...) and will amplify the two last mentioned points because the ratio allocated to the memory access will diminish.

Figure 3.1 shows the link between the clock frequency, the pipeline depth and the performance of scalar processors. The key message is that we can compensate the extra cycle delays due to deeper pipelines and even improve the performance of five-stage architecture (line (1) in figure 3.1) thanks to a higher clock frequency.
To achieve this purpose, a threshold frequency must be reached to justify the use of a more sophisticated architecture (like $f_{1,2}$ and $f_{1,3}$). The problem to solve is how to increase the frequency. To do so, ARC and MIPS have well balanced the logic within

new stages to reduce the critical path of the CPU (see paragraph 2.4.2) and add a fraction of the CPU clock cycle to the memory access time. As a result, a full clock cycle is allocated to obtain the data at the memory output that leads to increase the clock frequency (denoted as $f_{2_{sat}}$ in figure 3.1).
The proposed DSR architecture goes further in the sense that about 1.8 CPU cycle is given to access the primary data memory. Hence, the saturation frequency increases because the primary memory critical path has been removed or alleviated, giving more chance to be limited by the CPU frequency instead of the cache one. When such is the case, a bigger, and then slower, L1 data cache can be used: the size limit being the one that has the memory access time equal to the CPU saturation frequency ($f_{3_{sat}}$). This may imply less data cache misses and then contributes to increase the DSR performance—modelled by the vertical translation arrow of line (3) in figure 3.1.

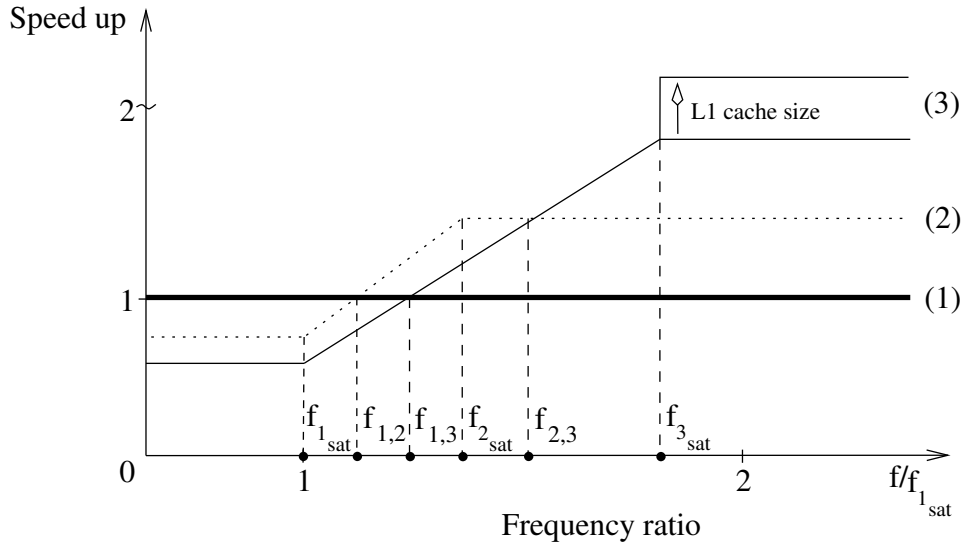

Figure 3.1: Expected Gain of DSR Compared to Scalar Architectures: $f_{i,j}$ is the threshold frequency that must be reach to pass from the architecture (i) to a more sophisticated architecture (j). (1) is the reference performance achieved by a five-stage processor like the MIPS R3000 architecture where the maximum frequency peaks at about 350 MHz ($f_{1_{sat}}$) in a 0.13μm High-performance process (2) represents the gain of performance brought by a deeper pipeline like the last MIPS32 24K over a five-stage processor where the higher reachable frequency is $f_{2_{sat}}$ (3) corresponds to the gain of performance brought by the DSR architecture compared to (1) where the maximum frequency is $f_{3_{sat}}$.

The threshold frequencies and the speed up are derived from the following formulas, where CPI stands for Cycle Per Instruction:

$$f_{i,j} \quad = \frac{CPI_j}{CPI_i} * f_{i_{sat}} \quad with \quad i,j \in \mathbb{N} \quad and \quad 3 \geqslant j > i \geqslant 1 \tag{3.1}$$

$$\begin{array}{lll} speedup_j & = \frac{CPI_1}{CPI_j} & when \quad f_j \leqslant f_{1_{sat}} \\ speedup_j & = \frac{f_j}{CPI_j} * \frac{CPI_1}{f_{1_{sat}}} & when \quad f_j > f_{1_{sat}} \\ where & 1 \equiv 5\text{-}stage\ architecture, & j \in [2,3] \end{array} \tag{3.2}$$

As long as the frequency is inferior to the saturation frequency of a five-stage processor (L1 caches included) $f_{1_{sat}}$, a five-stage architecture obtains the best performance. Moreover, it remains the best until the $f_{1,2}$ frequency be reached because deeper pipelines (lines (2) and (3)) need a surplus of frequency to compensate the extra wasted clock cycles induced by their numerous stages. From 0 to $f_{2,3}$, lines (2) and (3) are not superimposed because DSR is an architecture we have elaborated and implemented (see chapter 5), focusing our effort on the memory access issue and not on already known techniques like non-blocking loads, for instance. That the reason why the MIPS32 24K has better performance than DSR-HP until $f_{2,3}$, once its frequency has saturated ($f_{2_{sat}}$). But it is obvious that the proposed memory architecture could be applied to last generation of scalar deep pipelines and should improve their performance.

Furthermore, the gain gap beyond the $f_{3_{sat}}$ frequency will continue to grow for each future technology nodes due to wire delays that are not taken into account in current architectures.

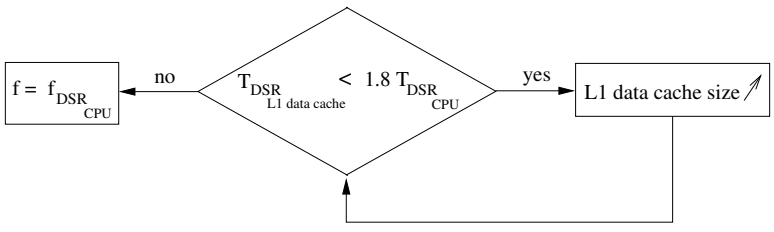

Figure 3.2: DSR's Algorithm to enhance the L1 data cache capacity: this figure illustrates the gain obtained in figure 3.1 for the line (3) without increasing the CPU frequency. Indeed, the gain resides in the time the memory has to return its data to the core: if time remains, one can chose a bigger, and thus slower, memory. High-density caches may be used.

Figure 3.2 shows how to take advantage of the double cycle memory access to increase the performance: while the L1 data cache does not limit the CPU frequency, increase its size and/or use High-density caches.

Whereas previous studies mainly focused on reducing the processor-memory gap in terms of cycles by working on the memory hierarchy, on-chip caches, processor-DRAM integration, on-chip SRAM or SRAM technology process, DSR tackles the L1 cache memory access time that plays a fundamental role in the processor-memory gap [40]. Thus, the DSR-HP architecture must lead to high performance thanks to its architecture, as described in chapter 4, because is planned to allow processors to reach frequencies that we can expect them to reach by not being stuck by the memory technology even for large cache capacity.

Chapter 4

The DSR Architectures

4.1 Introduction

This chapter defines a set of DSR architectures and describes how the microprocessors behave but without specifying how it should be built: the key point resides in the interface between the processor and its closest memory ressources. Furthermore, because the added value of the proposed architectures does not reside in the memory hierarchy, in the treatment of exceptions, on debug functionality nor in I/O system interfaces, all these points are standard, and will not be detailed in this work. DSR is based on a *Harvard* architecture, i.e., two separate physical memories and each with they own address and data buses, which allows two simultaneous memory accesses: one for fetching instructions and the other available to fetch operands.

4.2 The Pipeline

The DSR architecture is a single pipeline made up of eight stages, treats instructions in-order and performs, at most, one instruction per cycle. A dynamic branch predictor scans the second instruction of bundles where a *bundle* is a pair of instructions loaded every memory clock cycle (equivalent to two CPU clock cycles) from the L1 instruction cache. The stages mainly execute the following tasks as resumed in figures 4.1 and 4.2:

- IP: Instruction Pre-fetch stage
    - I-cache tag/data arrays accessed
    - Branch Target Instruction Cache accessed

- IF: Instruction Fetch stage
  - Detect I-cache hit
  - Instruction Buffer
  - Program counter incrementation
- BP: Branch Prediction stage
  - Program counter selection
- DF: Decoding
  - Register File access
  - Instruction decoding
  - Branch target address computation
  - Branch target instruction anticipation
- DS: early Decoding of the Second instruction of a bundle
  - Instruction decoding
  - Branch target address computation
- AG: data cache Address Generation stage
- EXE: Execute stage
  - Arithmetic and logic operations
  - Branch Resolution
  - Jump target address computation
- MF: First stage of data cache access
  - Start data cache access
  - Data cache hit detection
  - Storing of a store data
  - Storing of a store address
- MS: Second stage of data cache access
  - Complete data cache access
  - Load align
- SYNC: Synchronization stage
  - deal with memory instructions. Bypassed stage when a generated data memory address can be provided to the data cache at the end of the AG stage. Otherwise the computed address is stored during one CPU clock cycle before the memory access.
  - Data cache hit detection
  - Start data cache access for synchronized store instruction

- M1: single CPU clock cycle delay stage taken by non-memory instructions to ensure that these instructions will pass through the same number of stages than the memory ones. Thus, each stage is accessed once at a time.
- M2: same as M1
- NO: single CPU clock cycle delay stage taken by non-memory instructions and memory instructions that have bypassed the SYNC stage to ensure No Overlap between stages.
- WB: Writeback
    - Register file update

The fact that branches are predicted allows to increase the CPU frequency of the MIPS32 24K up to 625 MHz compared to the 330 MHz of the MIPS32 4Kc (see table 2.1) by cutting the path that returns data from the ALU to the ID stage to perform branch decision. Indeed, register comparison is made later in the pipeline, in the EX stage, alleviating the ALU-ID five-stage forwarding critical path. The same remark is valid for the DSR in the sense that the register comparison is always made in the EXE stage, independently of the position of the branch within the bundle (details in subsection 5.2.3).

An important rule of *thumb* is that two clock domains cohabit: one for the bare CPU and the other twice slower with the same phase, dedicated to the L1 memories. This peculiarity allows to extend the memory accesses upon two stages instead of one. Moreover, the memories are activated on the rising edge of the memory clock cycle.

As detailed in figures 4.1 and 4.2, three types of instructions exist: memory instructions, *branch* and *jump* instructions and all the others. Except for the latter category, the existence of an instruction through various stages depends both of the position of the instruction within the bundle (IP or IF) and of the instruction type. So a total of five different pipeline ways exist.

Many factors have an effect on the on-chip memory access time (clock-to-Q output time, T_{cq}) such as the memory capacity, its organization (associativity, line size, column-multiplexing factor), number and type of ports, the fabrication process and the technology. The DSR architecture alleviates the effects of any of these parameters on the memory access time to ensure high performance. However, the L1 memory interfaces lead to two different architectures: 32-bit[1] dual port L1 memories for high performance with the DSR-HP architecture, and a 32-bit single port data memory and

[1]In DSR, the word size is arbitrary. A requirement for DSR-HD is to have an instruction memory port size the double of instruction word size.

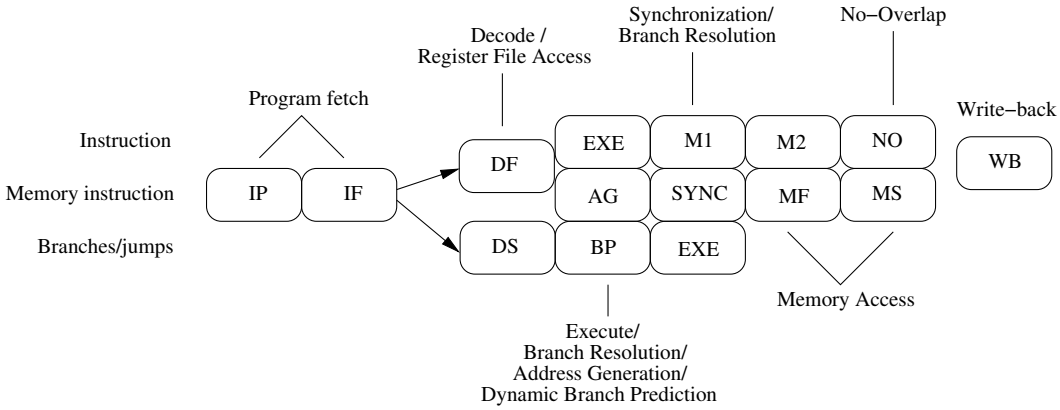

Figure 4.1: DSR Microarchitecture Block Diagram (IP): it is a unique pipeline with eight-stages that executes instructions in-order. A memory instruction located in the second instruction slot of the bundle (IP) is not synchronized with the L1 data cache and then goes from IP to WB via the DF and SYNC stages. The *branch* and *jump* instructions are decoded early (DS) and the target address evaluated to access the L1 instruction cache at the same moment that a *branch* in IP would do it: it goes from IP to EXE via the DS and BP stages. The third category of instructions goes from IP to WB via the DF and NO stages.

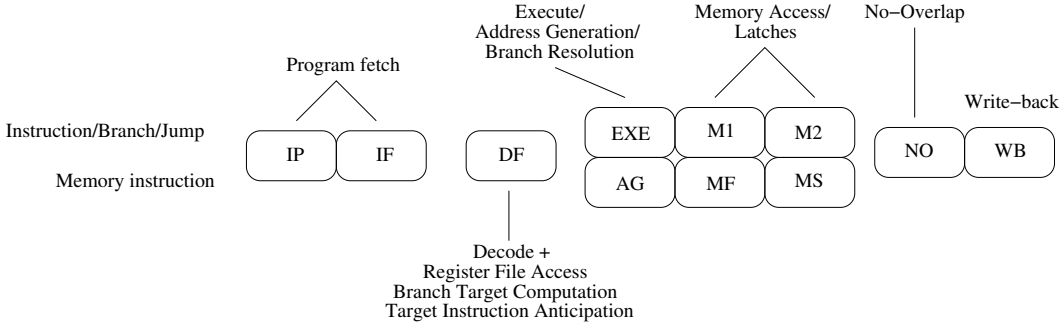

Figure 4.2: DSR Microarchitecture Block Diagram (IF): it is a unique pipeline with eight-stages that executes instructions in-order. A memory instruction located in the first instruction of the bundle (IF) is well synchronized with the L1 data cache and goes from IF to WB via the AG stage. All the others take the upper path, from IF to WB via the EXE stage.

a 64-bit single port instruction memory induce the DSR-HD architecture for low cost applications.

It should be noted is that when we refer to *high performance*, it is still in comparison to scalar processors. Furthermore, it is relative to the DSR architecture itself in the sense that DSR is performing due to the clock frequency that can be reached (primary on-chip memories included). The term of HP (respectively HD) has been

added because DSR can reduce the impact of a deep pipeline in a wasted cycles viewpoint (respectively reduce the silicon area used).

So, what follows illustrates the effects of a chosen architecture (HP or HD) on the performance.

4.3 A High Performance Architecture: DSR-HP

The High-performance architecture resides in the accessibility of the primary instruction cache and the availability of the primary data cache. These caches are fully accessible and available for the following reasons: first, DSR-HP is a Harvard architecture, i.e., two different memories are allocated for the data and instruction caches with their own buses. Then, both caches are 32-bit dual port memories. Consequently, a branch target instruction is directly loaded and two data memory access may occur at the same time as shown in figures 4.3 and 4.4.

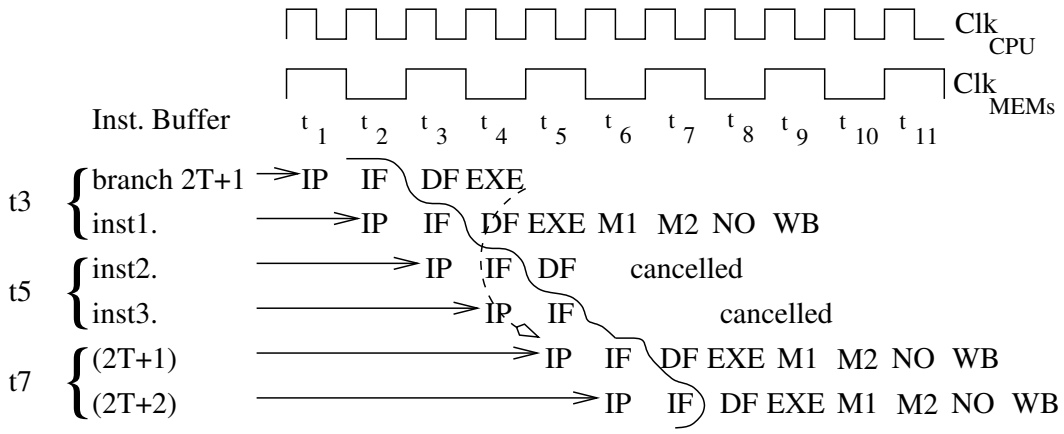

Figure 4.3: Target Instruction Fetching without Penalty: due to the 32-bit dual port instruction memory, DSR can directly access any even or odd target instruction within that memory and the next one. We can note that instructions *ints2.* and *inst3.* are cancelled—resulting a lost of two clock cycles—because *(2T+1)* and *(2T+2)* were not in the BTIC (see subsection 5.2.3). In this example, a single *branch delay slot* is considered, filled by an instruction, *inst.1*

4.4 A High Density Architecture: DSR-HD

The DSR-HD architecture is designed to occupy a minimum of silicon area when performance is not mandatory. To do so, the primary memory area is reduced by

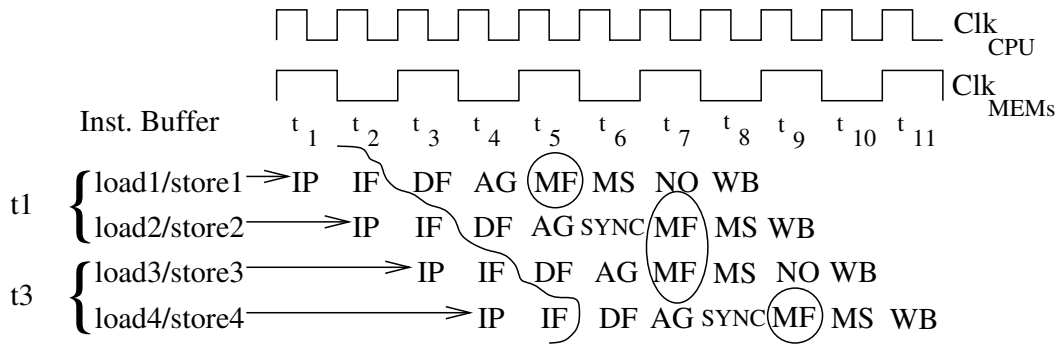

Figure 4.4: Successive L1 Data Memory Accesses: due to the 32-bit dual port data memory, DSR can access that memory at any CPU time cycle. This occurs when memory instructions are present both in the second instruction slot of a bundle and in the first one of the next bundle: *load2/store2* is 1-cycle delayed and thus synchronized with the data memory whereas *load3/store3* bypasses the *SYNC* stage to remain synchronized with the same data memory. Consequently, the data memory is accessed twice at t_7. Independent instructions are considered in this example.

replacing dual port memories by single port ones. Indeed, single port memories occupy less silicon area that their dual port counterparts as shown in table 4.1. Consequently, DSR-HD uses a 64-bit single port instruction memory and a 32-bit single port data memory.

SRAMs	**CL013LV-HS**			**CL013G-HD**			**CL013LV-HD**		
	8KB	**16KB**	**32KB**	**8KB**	**16KB**	**32KB**	**8KB**	**16KB**	**32KB**
64b Single Port	36.5	3.7	3.4	19.2	6.1	7.5	6.5	7.7	6.4
32b Dual Port	151.6	113.3	108.8	90.8	86.7	94.2	97.3	100.7	92.4

Table 4.1: Virage Logic's SRAM Silicon Area: to keep the Virage Logic's [66] and TSMC's [64] information as confidential, the above silicon area results were normalized as of their corresponding 32-bit single port compiled SRAMs, e.g., an 8KB 64b single port SRAM occupies 12,8% more silicon area than its 32-bit single port compiled SRAM countepart, for a same High-speed CL013LV process. All the SRAMs were compiled with the TSMC's High-speed CL013LV (130nm Low Voltage process), High-density CL013G (130nm standard process) and High-density CL013LV processes.

Furthermore, table 4.2 shows that DSR-HD is a real High-density solution because the silicon area involved in primary memories is close to the one necessary for a five-stage processor with 32-bit single port primary memories (between 3.2% to 3.9% for the High-density CL013LV process). Moreover, the MIPS32 24K needs twice more surface for its L1 memories than DSR-HD.

CPU cores	CL013LV-HS			CL013G-HD			CL013LV-HD		
	8KB	16KB	32KB	8KB	16KB	32KB	8KB	16KB	32KB
MIPS32 24K	36.5	3.7	3.4	19.2	6.1	7.5	6.5	7.7	6.4
DSR-HD	18.3	1.9	1.7	9.6	3.2	3.8	3.2	3.9	3.2

Table 4.2: MIPS R3000, MIPS32 24K and DSR-HD's L1 Memory System Area: 32b single port data and instruction memories of a 32b five-stage processor are used as the area reference. The MIPS32 24k uses 64b single port data and instruction memories whereas DSR-HD is based on both a 64b single port instruction memory and 32b single port data memory.

Knowing that the primary memories occupy the main part of the processor chip, the previous results are even more relevant. For instance, two 8KB single port L1 memories of a five-stage processor like the ARC 600 [17] represent roughly five times the CPU area: this core does not contain a *Memory Management Unit* (MMU) and is made up of 27,000 gates [18].

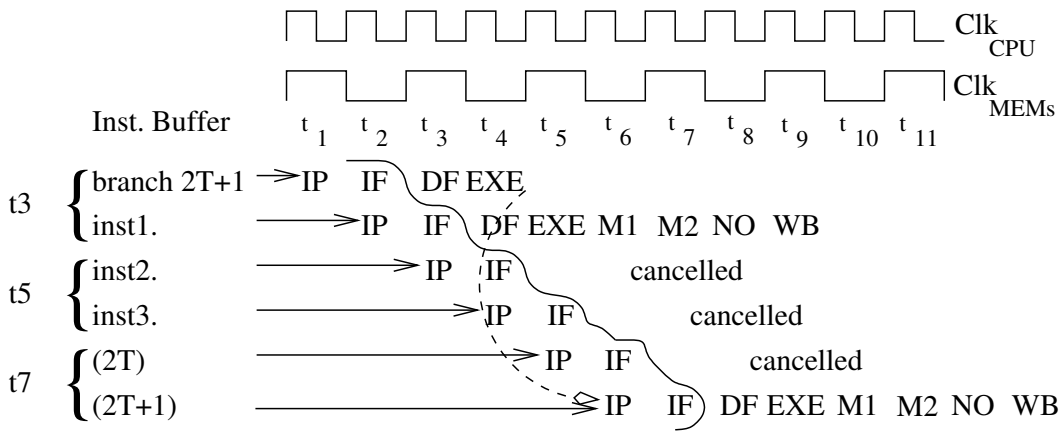

Figure 4.5: DSR-HD's Performance on a Target Instruction Fetching: due to the 64-bit single port instruction memory, DSR cannot directly reach the concerned target instruction and its following instruction whenever the target instruction address is not 64-bit word aligned. Thus, at t_5, a bundle is loaded that contains only one useful instruction instead of two, as its the case for DSR-HP (see figure 4.3). Consequently, the undesirable instruction *(2T)* is removed from the instruction buffer. We can note that instructions *ints2.* and *inst3.* are cancelled—resulting a lost of two more clock cycles—because *(2T+1)* and *(2T+2)* were not in the BTIC (see subsection 5.2.3). In this example, a single *branch delay slot* is considered, filled by *inst.1*

The impact of a High-density DSR architecture on the performance is shown in figures 4.5 and 4.6 where the access to an instruction or a data is not always optimum due to an extra clock cycle delay.

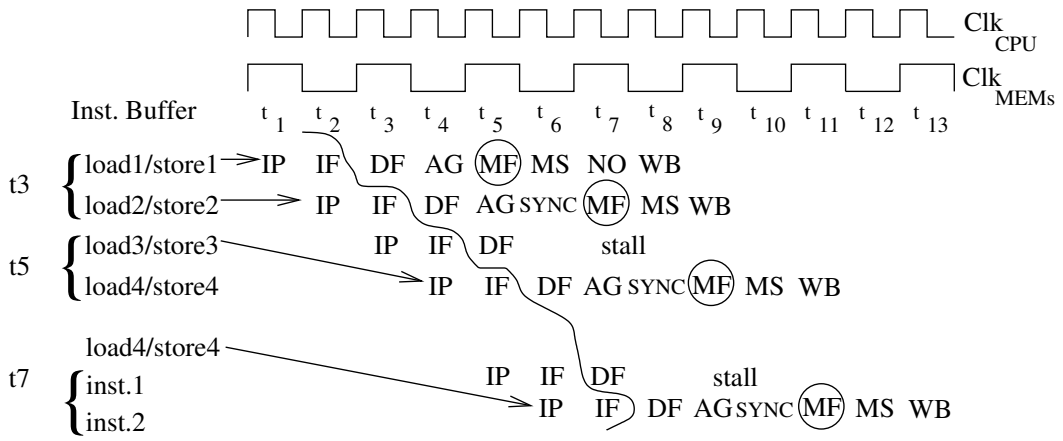

Figure 4.6: DSR-HD Data Memory Access Delay: whenever two consecutive memory instructions that do not belong to a same bundle occur, a stall cycle is required after the first one is performed because the data memory ressource is then busy due to its unique port. In this example, independent instructions are considered.

4.5 A Low Power Architecture: DSR-LP

4.5.1 Motivation for Low-power Design

The number of million of CMOS transistors that can be packed in a given silicon area increases with the advances of deep-submicron silicon technology. As a consequence, the heat that a (High-performance) circuit dissipates drastically increases: the power density of High-performance circuit is planned to be the same as the sun by the end of 2010 [45]. So, power consumption is a concern for integrated circuit makers.

Furthermore, the power budget of SoC is limited, especially for embedded circuits where the battery life time and reliability is crucial. Indeed, electromigration may affect metal pitch and routability, and temperature has an effect on electrical performance.

Low-power designs might use slower memory to save silicon and reduce power consumption [18]. Conversely, high-performance designs might have large on-chip memories that cannot deliver single-cycle latency at high clock speeds.

4.5.2 Power Dissipation in CMOS Circuit

Dynamic power consumption depends on capacitance, frequency and voltage ($P=cv^2f$) where voltage has a quadratic effect which implies that nothing will lower power like

the voltage. With deep sub-micron technologies, static power consumption becomes the main important factor.

4.5.3 Voltage Scaling

The DSR-LP architecture is the same as the DSR-HD one but tackles the L1 memory power consumption with techniques like *Clustered Voltage Scaling* (CVS) [8][19][26] and *Dynamic Voltage Scaling* (DVS) [52] for energy reduction. The goal here, is to use the surplus of time one has to access the L1 memories in order to reduce the primary memory power supply at a level that implies a same frequency as the bare CPU. In other words, the idea here is to run the DSR CPU core with the required power supply and feed the L1 caches with a reduce Vdd because they do not need to run at the maximum frequency but at a half of the CPU speed.

The principle is as follows: let us call τ the five-stage processor critical timing path. It depends, among other parameters, on the power supply, Vdd, as described in formula 4.1 which has a great influence on τ and the main (quadratic) influence on power consumption as mentioned in formula 4.2 [41]. DSR implies that the data memory access is no longer the critical path, excepted for very large L1 memories (see figure 3.2), due to its two memory access stages. In any case, a frequency ratio roughly equal to 1.8 exits between the CPU and the L1 memories[2], leading to $\tau_{mem} = 1.8 * \tau$. Hence, a lower Vdd_{mem} power supply can be elaborated to make the previous equality true and decrease the energy consumption.

$$\tau = K * C_L \frac{Vdd}{(Vdd - Vt)^{\alpha}} \tag{4.1}$$

$$P = C * V^2 * f \tag{4.2}$$

4.6 Conclusion

The DSR embedded processors are eight-stage scalar pipelines that use two clock domains: one for the bare core and the other—twice slower—for the primary memories.

Three architectures derive from DSR:

- DSR-HP, dedicated to applications that required a high computation power:

[2] 20% of the CPU clock cycle being reserved to return the data from the data memory to the core [47].

thanks to its 32-bit dual port instruction and data memories, all the instructions are reachable at any cycle time and 32-bit data can be loaded or stored every CPU clock cycle (high flexibility). Furthermore, the maximum frequency is higher than a traditional scalar processor but it occupies more silicon area than a processor made up of single pipeline.

- DSR-HD, dedicated to applications where the cost is the main concern. Indeed, it uses small primary memories: a 64-bit single port instruction memory and a 32-bit single port data memory. The maximum frequency may be higher than for DSR-HP because dual port are slower. However, DSR-HP should offer a better CPI than DR-HD due to its high flexibility.

- DSR-LP, dedicated to ultra low power applications: it is very similar to DSR-HD but the power supply of the primary memories can be reduced and adjusted to the frequency. Thus, the power consumption is also reduced.

The rest of this document and especially chapter 7 focus on the DSR-HD architecture that will appear to be quite efficient when both performance and surface are taken into account.

Chapter 5

The DSR Microarchitecture

5.1 Introduction

This chapter defines the microarchitecture peculiarities of an implemented DSR-HP version where *caches* will refer to the first level on-chip memory that receive addresses directly from the CPU. Furthermore, the instructions will be divided into three categories: the *memory* instructions, the *branch/jump* instructions and the *non-memory* instructions that will refer to the instructions that remains (coprocessor instructions excepted).

Then, a focus on the influence of a deep pipeline on the latency of peculiar instructions is discussed.

A RTL implementation of a simplified version of DSR is detailed to show the frequency that may be reach.

Finally, a study on the benefit of the DSR architecture on the power consumption is achieved.

5.2 The Pipeline Stages

5.2.1 The Fetch Unit

For both High-performance and High-density, the DSR architecture stores the fetched instructions in a six-entry instruction buffer (figure 5.1) in order to support *load* stalls (see figures 4.6 and 5.3). Instructions are stored and shifted deeper in the buffer whenever a stall occurs, otherwise the buffer is bypassed to directly feed the IF and IP stages.

In figure 5.1, the $S_{i,j}$ represent the pair of IR_i and IR_j registers that feeds the IP and IF stages, respectively. Figure 5.2 models the Finite State Machine (FSM) that monitors the travel of the instructions from the memory instruction to the IF and IP stages through the IR registers and where $S_{1,0}$ is the normal case, i.e., no stall occurred and the buffer is bypassed.

The chronograms of figures 5.3 and 5.4 show how the instructions behave in the instruction buffer.

5.2.2 Decode Stage

The instruction in the IF stage is decoded and the instruction in the IP stage is predecoded in parallel by the DF and DS decoding stages, respectively. First, the goal is to detect a `branch` or a `jump` in the IP stage in order to access the instruction memory one cycle earlier, as if the *branch* was in the first instruction slot of the bundle (see figure 5.5). Consequently, both computed ((a)) or predicted ((b)) target addresses can be provided to the instruction memory at t_5.

A register file made up of thirty two 32-bit registers is accessed during the DF stage through two read ports.

Furthermore, a hardware data dependency is achieved during the DF stage between a register updated by a *load* instruction performed during the second half of the memory clock period and the third instruction that follows it as described in figure 5.6. Whenever a dependency is detected a 1-cycle *stall* operates.

5.2.3 The Branch Unit

5.2.4 The Branch Predictor

Due to the memory clock that is twice slower than the CPU one but with the same phase, the target branch address cannot be given to the memory instruction until the EXE stage (see (a) of figure 5.5): so, the *branch* has time to be decoded and to read registers during the DF stage, to finally resolve its condition in the EXE stage. Consequently, no branch predictor is required in that case.

However, it is not the same story when the *branch* instruction belongs to the second instruction slot of the bundle (see (b) of figure 5.5): indeed, if the same principle was used, the address would be computed at t_5 and available at t_6 but could not be send until the next cycle (t_7). To avoid this drawback, a branch predictor estimates

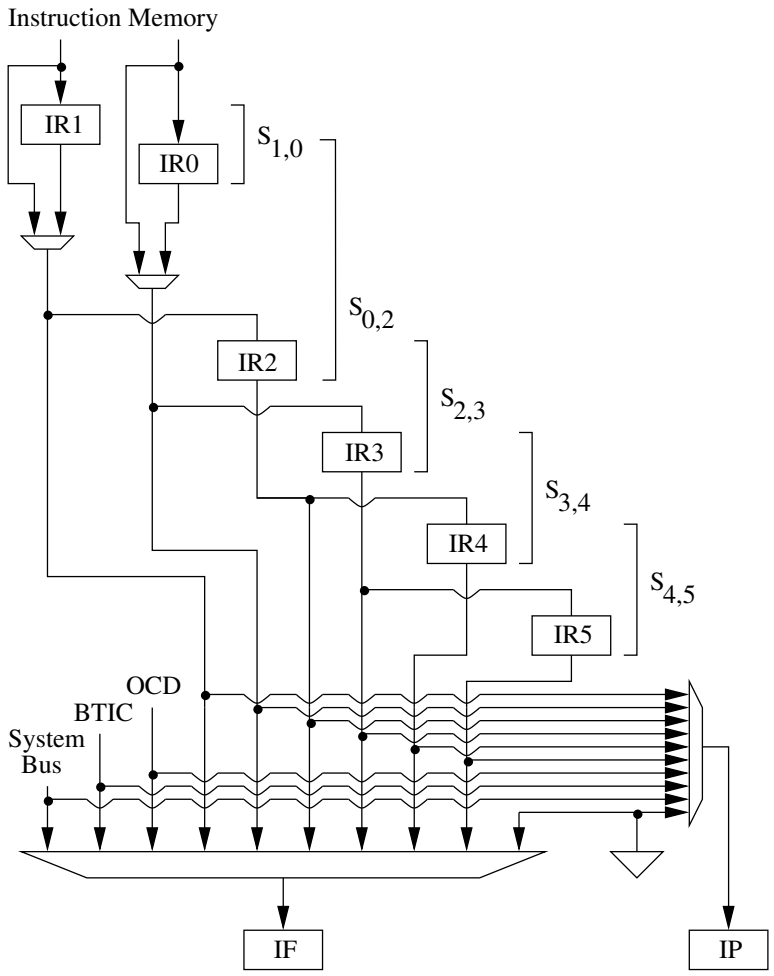

Figure 5.1: DSR's Six-entry Instruction Buffer: the two first stages of the pipeline—IP and IF—can be fed by the memory instruction itself, the six-entry instruction buffer (IR0-IR5 registers), the Branch Target Instruction Cache (BTIC), the external bus system, the On-Chip-Debug (OCD) interface or the *stall* mechanism which introduces NOPs. One can note that half the instructions never pass by IP and thus, stays a cycle of less within the pipeline than the second half: this is due to the fact that two instructions are fetched per memory clock cycle and hence, one is directly treated while the other waits for its turn one CPU clock cycle.

the decision to access the instruction memory at t_5: the branch is decoded in the DS stage, simultaneously with the previous one. Then, the BP stage choses to take the *branch* or not, without reading the registers involved in the condition to avoid data dependency with the instruction that is being executed. Finally, one CPU cycle later, the EXE stage confirms or cancels the decision taken by the branch predictor.

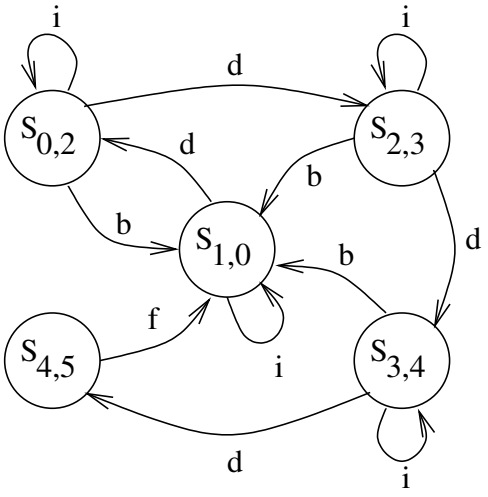

Figure 5.2: Instruction Buffer's Finite State Machine: $S_{1,0}$ is the normal mode where the buffer is bypassed. $S_{i,j}$ means that the IRi and IRj registers provide their instruction to the IP and IF stages, respectively and where the instruction buffer contains *j+1* instructions. The buffer fills up of instructions each time a *stall* is required and the FSM moves to the next S step (d transition). A taken *branch/jump* instruction empties the buffer and then, the FSM goes to the normal $S_{1,0}$ mode (b transition). When the instruction located in IR3 is shifted to IR5 ($S_{4,5}$ step), the two instructions that follows IR3—IR0 and IR2—are reloaded from the instruction memory and the buffer is cleaned (f transition).

The branch predictor uses both dynamic and static prediction techniques.

5.2.4.1 Dynamic Branch Predictor

It is based on a BTIC that stores a taken branch address, its corresponding target instruction and the two next ones. The four last taken branches are stored in this cache which is updated each time a new taken branch occurs by replacing the oldest branch in the buffer[1]. The BTIC is addressed at the same time that the instruction memory by two same instruction addresses.

If the look up matches, the *branch* is estimated as taken. The BTIC will also acts as an *anticipator* in the sense that is no longer necessary to address the instruction memory because the next three instructions are already available because they are located in the cache. This eliminates waste cycles mentioned in subsection 5.2.5 where up to three cycles can be saved in the case of mispredicted branches (see tables 5.1 and 5.2).

[1]Simulations using EEMBC benchmarks have shown that a buffer depth higher than four did not improve the performance.

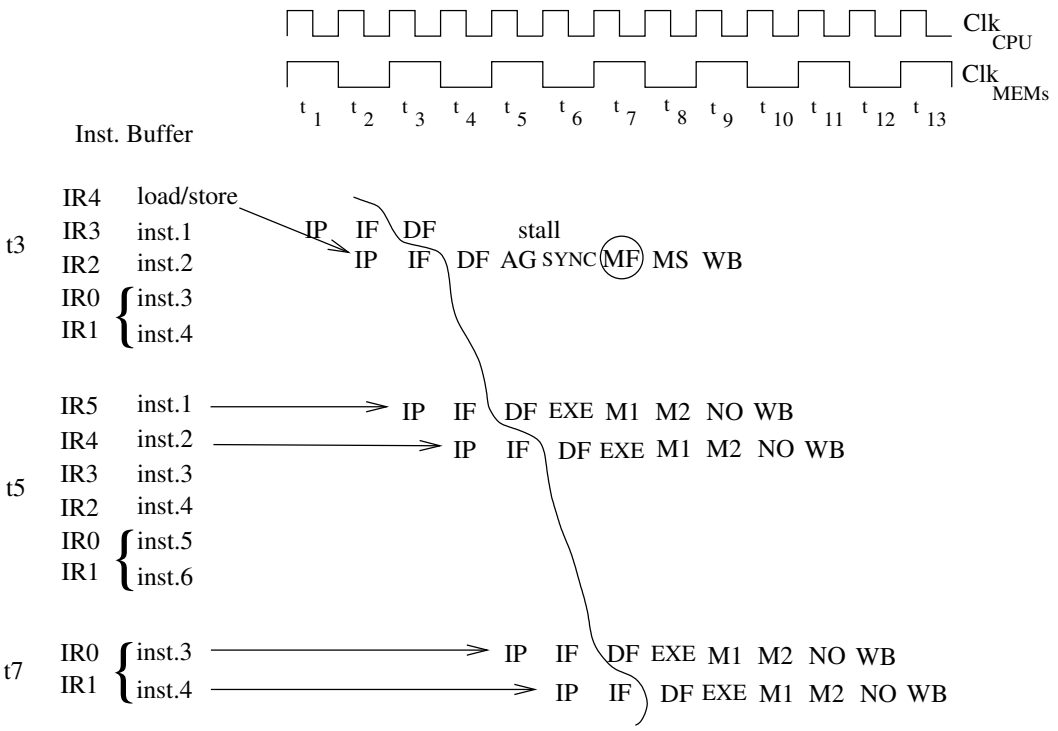

Figure 5.3: Update of the instruction buffer when its full: the fetch unit can support up to four *stalls* before reseting the instruction buffer, the last one occurring at t_3. Then, the content of IR3 will be stored in IR5 and the *inst.3* and *inst.4* instructions will be recalled at t_5. As a result, the instruction is emptied and filled only with the *inst.3* and *inst.4* instructions at t_7.

5.2.4.2 Static Branch Predictor

When the *branch* to load is not in the BTIC, the hand is given to the static branch predictor that will apply a backward taken algorithm, i.e., the target address is computed during the DS stage and compared with the current program counter during the BP stage to chose the *branch* as taken if it has a smaller address. It is based on the fact that the loops verify this rule.

5.2.5 Target Instruction Fetching Delay

So, DSR needs two or three CPU clock cycles before reaching the target instruction[2]—a *branch delay slot* equals to one, in MIPS's terms, plus two interlocks, at most, when needed—depending on the location of the branch in the bundle. Indeed, a delay slot

[2]a fourth delay cycle, implemented as an interlock, may be required for DSR-HD as illustrated in figure 4.5

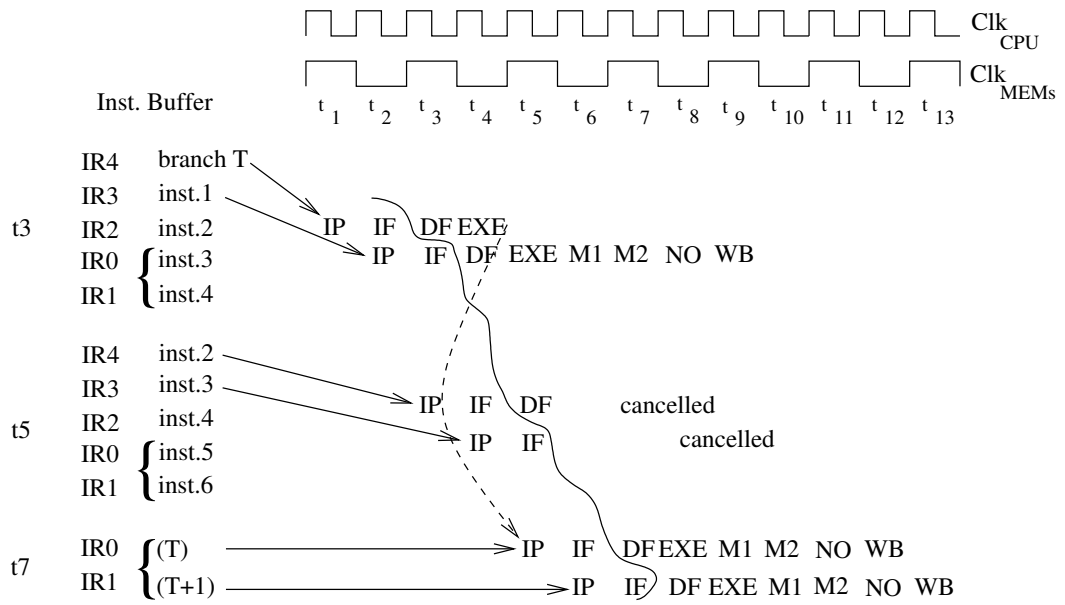

Figure 5.4: Instruction buffer update due to a taken branch instruction: whenever a *branch* is taken, the instructions that remain within the instruction buffer are no more useful, excepted the one in the *branch delay slot* (inst.1). As a consequence, the buffer is emptied at t_7 and filled with target instructions.

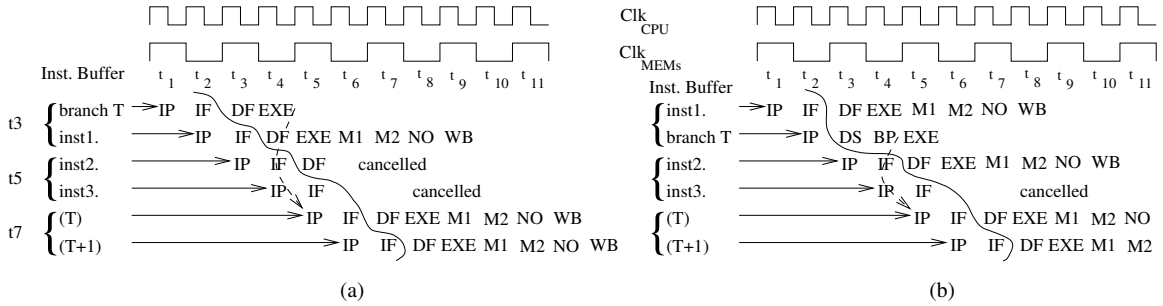

Figure 5.5: DSR's Decoding Stages: these two sequences treat a taken branch instruction but with the following difference: (a) the branch is in the first instruction slot of the bundle and *inst.1* fills the *branch delay slot*. The branch is decoded in DF as a standard instruction. (b) the branch is in the second instruction slot of the fetched bundle and *inst.2* fills the *branch delay slot*. The branch decoding is 1-cycle anticipated and performs in parallel with the decoding of *inst.1*. In these examples, a single *branch delay slot* is considered, filled by either by *inst.1* ((a)) or *inst.2* ((b))

---

and one stall are sufficient whenever the *branch* occupies the second slot of the bundle 5.5. A fourth stall cycle is required when a *not taken branch* is mispredicted (see table 5.1).

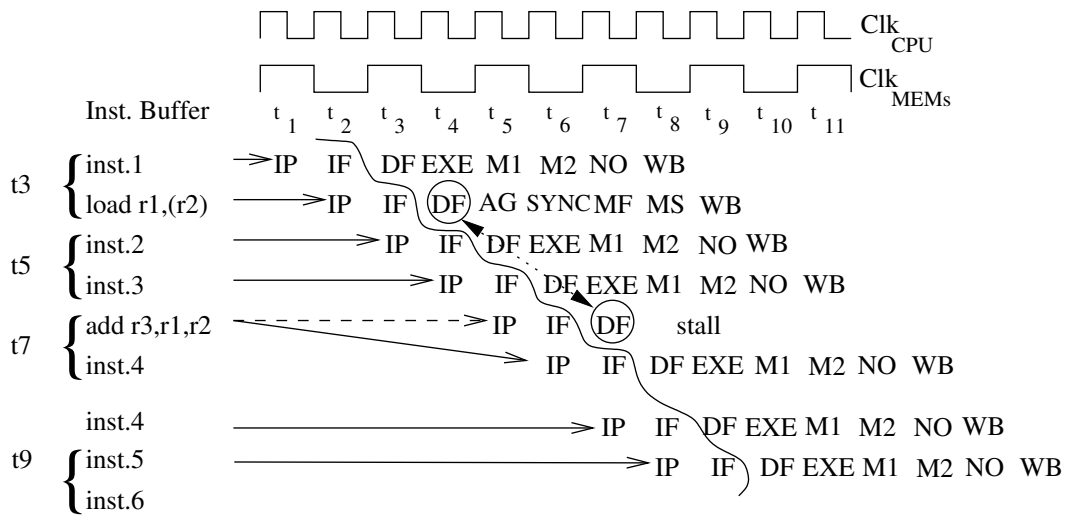

Figure 5.6: Data Dependency Detection: during the DF stage of the addition, at t_5 an operand dependency is detected with the previous *load* instruction on register *r2*, which implies to stall the pipeline during one CPU clock cycle.

Even if more cycles are required to get the target instruction compared to the MIPS's single cycle *branch delay slot*, the critical path that goes from the input of the ALU to instruction memory address port is removed. Indeed, a MIPS five-stage processor without branch predictor, evaluates the branch condition during its decoding stage and hence, needs the values of the involved registers that can be either at the ALU or the data memory output, instead of the register file.

5.2.6 The Program Counter Unit

Once a bundle is fetched, potential new program addresses are systematically computed from the two loaded instructions: This is possible because we are faced to RISC instructions that have the peculiarity to have a fix instruction format (the base registers and offsets are always at the same place within an instruction, depending on its type). To do so, seven 16-bit adders[3] calculate in parallel, each memory clock cycle, seven address values. Indeed, the program counter is:

- incremented by two when instructions are executed sequentially.
- added to a branch offset in case a taken *branch* is not misspredicted and its target instruction not anticipated (see subsection 5.2.4.1).

[3]They can address up to 64K 32-bit instructions.

- incremented by one and added to a branch offset to fetch the instruction following the previous target instruction and thus, complete the bundle.
- incremented by one and added to a branch offset when a taken *branch* was predicted and its target instruction not anticipated (see figure (b) 5.5).
- incremented by two and added to a branch offset when a taken *branch* is performed in the first half of the memory clock cycle and its target instruction not anticipated (see figure (a) 5.5).
- incremented by three and added to a branch offset when a *branch* was chosen as not taken and was misspredicted (see table 5.1).
- incremented by four and added to a branch offset to fetch the instruction following the previous target instruction and thus, complete the bundle.

Furthermore, shifted immediate addresses of jumps are also generated. An exception with *relative jumps* which provide their target address from the content of a general purpose register. Consequently, to avoid data dependency, this address is given two cycles after the EXE stage, whenever a *relative jump* is performed in the second half of the memory clock cycle. All these generated addresses allow to ask the dual port instruction memory for two consecutive instructions to be executed.

5.2.7 Execution Stage

The EXE stage operates standard integer arithmetic and logic operations. Among other things, it is made up of a 32-bit adder, a shifter and a Multiply and Divide Unit (MDU) which is independent of the rest of the CPU with its own special output registers. Integer multiplication and division are relatively slow and take several cycles, depending on the implementation.

Branch's conditions are resolved in this stage.

5.2.8 Address Generator Stage

The AG stage operates at the same level that the EXE stage but compute memory addresses with a 16-bit adder: 32b memory of 256KB in size can be addressed. This small adder alleviates the address translation and propagation delay to primary memories mainly due to the fact that the addition is much faster than the EXE's one.

5.2.9 Data Memory Stages

One can distinguish four memory stages, M1, M2, MF and MS where M1 and M2 are dedicated to non-memory instructions. The EXE stage result is just latched and

transmitted to the M1 stage and then to the M2 stage to finally reach the NO stage, three CPU cycle later. These two stages allow to avoid resource conflict by ensuring that each stage is in use only once at a time.

The MF and MS stages are the first and second step of a L1 data memory access, since DSR extends the primary memory access upon two stages. Indeed, the memory address is given to the memory at MF that can handle up to two addresses, in case of a double access like in DSR-HP.

In a *load* case, once the data is present at the memory output data during the MS stage, data is then transmitted to the CPU core where two sign extensions and byte alignments may be performed by the end of the MS stage timing slot.

In a *store* case, the data is written to the L1 memory during the MS stage. Moreover, before writing the data into the data cache (if any), the processor must check whether the correct cache line resides in it. To do so, once the program address has been generated by the AG stage, a tag cache is read to determine if the *store* hits or misses the data cache while the datum and its memory reference are stored in buffers to access the data cache at the next store, which will be itself stored in the buffers as illustrated in figure 5.7 part (a): it shows how a *write* behaves both within the pipeline and in the write buffers: at t_5, tag checking with the current address, *(R1)*, is performed during the MF stage, and the datum, *R2*, and its memory reference are stored into two 32-bit registers, part of a Store Data Queue (SDQ) and of a Store Address Queue (SAQ), respectively. The following *store* instruction launches the access to the data memory at t_9 for the previous write and is itself store in the write buffer at t_8.

Three consecutive *store* instructions determine the maximum buffer size, and leads to a depth of three for both the local memory that holds the datum—the Store Data Queue (SDQ)—and the one that holds the corresponding memory reference—the Store Address Queue (SAQ)—as shown in figure 5.7 part (e).

5.2.10 The Eighth Stage

Up to now, all the instructions would cross the same number of stages, from the moment they are decoded to the WB stage. However, due to the fact that the primary memory clock frequency is twice slower than the CPU one, it may happen that the L1 data memory cannot be accessed directly after the AG stage because the generated address would arrive in the middle of the memory period. Typically, this occurs when a memory instruction is the decoded during the second half of the memory period (see

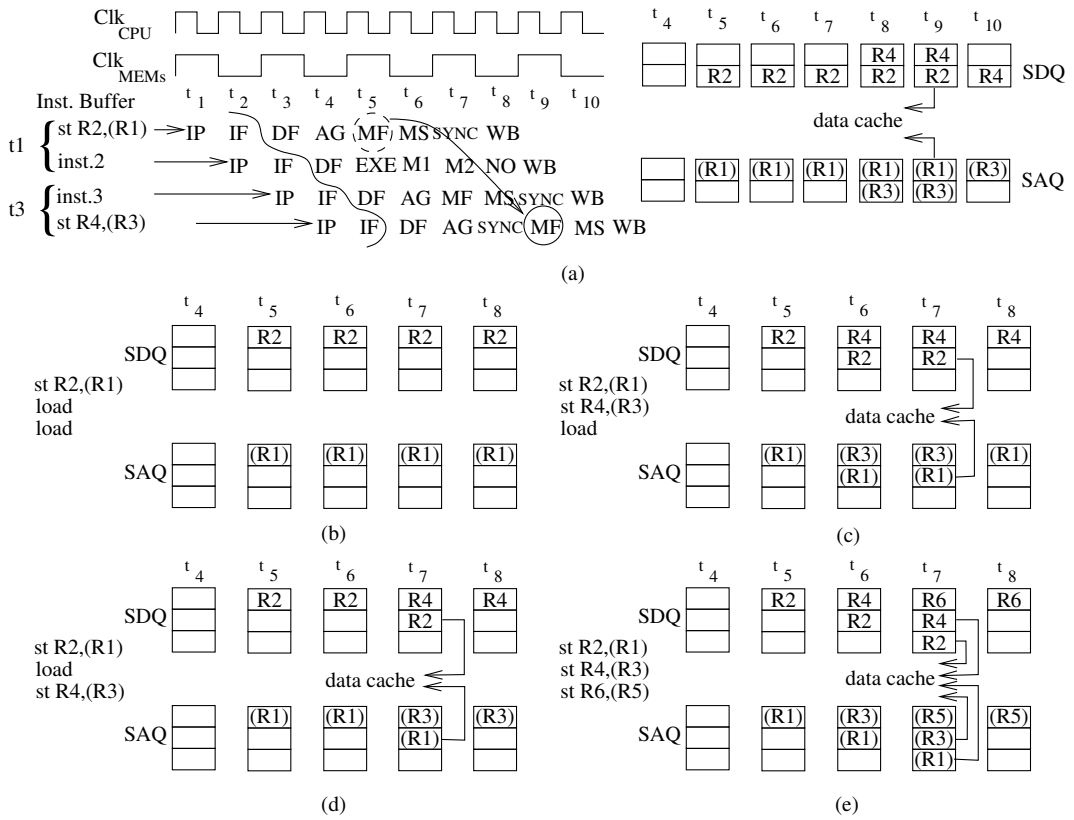

Figure 5.7: Store Data and Address Queues: the sequence of instructions in figure (b), (c), (d) and (e) are in the same time slot than the one of figure (a).in order to avoid extra load delay penalty, only a *store* instruction can be stored in SDQ and SAQ. A *store* instruction accesses the data cache during the hit detection cycle of the next write instruction.

figure 4.4).

As a consequence, such instructions must be synchronized with the L1 data memory. To do so, one more stage is introduced between the AG and MF stages to act as a 1-cycle buffer when necessary otherwise it is bypassed: the so-called synchronizer stage is named SYNC. This solution has the advantage to save a CPU clock cycle whenever a non-synchronized memory instruction has to be executed, unlike the Xtensa LX, with its seven stages, which interlocks the pipeline during one cycle. This architectural choice is even more relevant when one knows that memory instructions are the instructions the most frequently executed in programs: even if it varies with the program, roughly 30% of the total number of executed instructions are memory instructions [20].

So, statistically, half of the memory instructions cross the SYNC stage. As a result,

the second half pass through one stage of less which would lead to stage access conflicts. To solve this problem, the following rule is applied: all the memory instructions that do not take the SYNC stage before the L1 data memory access (before the MF stage) will take it after (after the MS stage). Figure 4.4 illustrates this rule and shows that this new stage is also in use once at a time.

Likewise, all non-memory instructions (branches and jumps excepted) would pass through one stage of less than the memory instructions. Similarly, a other stage has been introduced between the M2 and WB stages that acts as a buffer to tackle this issue that concerns only non-memory instructions: this regulator stage is named NO, which stands for *Non-Overlap*.

To summarize, except *branches* and *jumps*, all the instructions crosses six stages from the decoding to the WB stage, plus the time to fetch a pair of instructions that corresponds to the IP and IF stages. Thus, the resulting architecture is a single instruction issue, in-order eight-stage pipeline that keeps a simple register file (a write and two read ports).

5.2.11 Write Back Stage

Due to the pipeline structure, the WB stage is accessed one at a time and updates the thirty-two 32b register file through its single write port, if needed.

5.3 Multi-cycle Instructions

Increasing the pipeline length introduces wasted cycles due to data and control dependencies that are stressed, even for the High-performance version of DSR as we will see in this section. Typically, this phenomenon concerns *load*, *store*, *branch* and *jump* instructions that may have a CPI higher than one.

5.3.1 Branch Latency

What follows resumes and summarizes the *branch execution flows* discussed earlier in sections 4.3, 4.4 and 5.2.3. It shows up that up to four extra CPU cycles may be required to handle control flow instructions.

Six cases may occur:

- the *branch* instruction is in the first half of the memory clock period (IF stage) and can be either taken or not taken since no prediction is performed in this case.

- the *branch* instruction is in the second half of the memory clock period (IP stage). A prediction is made which induces four possible cases: the *branch* is well predicted taken or not taken, or it is misspredicted taken or not taken.

To well understand table 5.1, it is good to remember that a *branch* is followed by a *delay slot* which may be filled by a usefull instruction if the compiler found one (more details in subsection 5.2.5). Finally, the time that needs the CPU for fetching the target instruction (T) can be used to anticipate this requested instruction (*anticipated (T)*) and the following, if it has been detected in the BTIC.

	Taken		Not Taken	
IF	br ds anticipated T anticipated T+1		br i+1 i+2 i+3	
	miss	not miss	miss	not miss
IP	br i+1 i+2 stall stall	br ds anticipated T T or T+1 T+1 or T+2	br ds stall stall stall	br i+1 i+2 i+3 i+4

Table 5.1: DSR-HP's Branch Control Flows: by default, the instructions (i+j) that follow a *branch* instruction in the code, always starts to be executed and can be cancelled in case of a misspredicted, excepted the one within the *branch delay slot* (ds). For DSR-HD, an extra cycle could be required when a branch is taken because only the 32-bit target instruction (T) may be contained within the 64-bit instruction word.

Table 5.2 expresses the various possible flows described in table 5.1 in terms of *branch* CPI. It varies from one to five like the MIPS32 24K whereas the MIPS five-stage processor *branch* CPI goes from one to two. So, obviously, deeper is the pipeline higher is the number of potentially wasted clock cycles. The overall performance will be given once many factors like the frequency and programs will be take into account.

5.3.2 Jump Latency

This paragraph is the counterpart of the previous subsection but for *jump* instructions which are divided into two categories:

- *absolute* jumps (referenced as J) where the target address is contained in the instruction itself.
- *relative* jumps (referenced as JR) where the target address is contained in a general-purpose register that is pointed out by the instruction.

	Taken		Not Taken	
IF	min.: 1 max.: 4		min.: 1 max.: 1	
	miss	not miss	miss	not miss
IP	min.: 3 max.: 3	min.: 1 max.: 3	min.: 4 max.: 5	min.: 1 max.: 1

Table 5.2: DSR-HP's Branch CPI: many situations may occurs for the second instruction of a bundle (IP stage) because a branch predictor is used. For DSR-HD, the maximum value of a CPI corresponding to branch taken path may be increased by one because only the 32-bit target instruction (T) may be contained within the 64-bit instruction word, and then the remaining instruction of the loaded bundle can be useless.

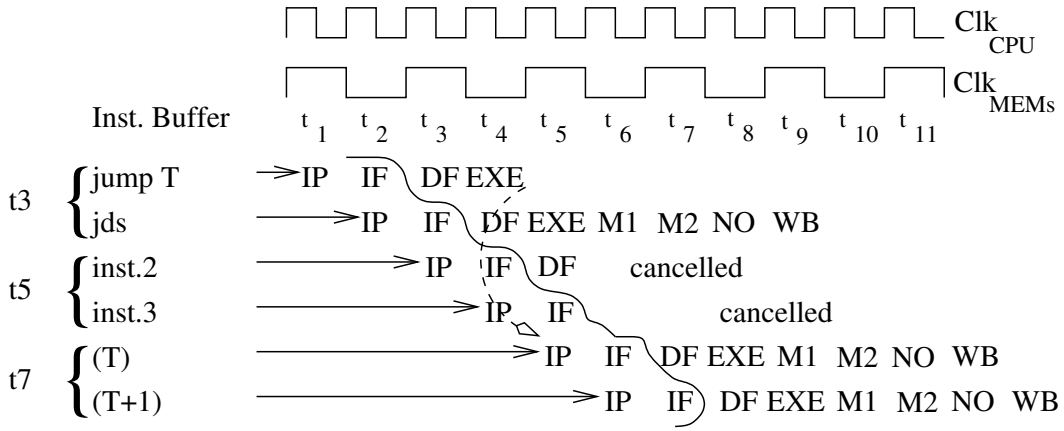

Figure 5.8: Target instruction latency induced by a *jump* instruction located in the IF stage: the target address, T, of all the *jump* instruction (J and JR) located in the IF stage are computed in the EXE stage and given to the instruction memory at t_5. As a result, the instruction within the *jump delay slot* (jds) is performed and the two next, *inst.2* and *inst.3* are undesirable and then are discarded. The target instruction arrives at t_7.

The target addresses of *jump* instructions are generated in the EXE stage as shown in figures 5.8 and 5.9. However, this rule has an exception: indeed, for performance reasons, the target address of *absolute jump* instructions, located in the IP stage are prematurely extracted in the DF stage in order to access the instruction memory at the end of this stage as illustrated in figure 5.10. This is feasible because the target address appears in the intruction as it is, and then no data dependency may exist.

One can also notice that *jump* instructions are not concerned by missprediction like *branches* are, because there are unconditional jumps.

Tables 5.3 (a) and (b) resume what preceeds in term of sequence of instructions

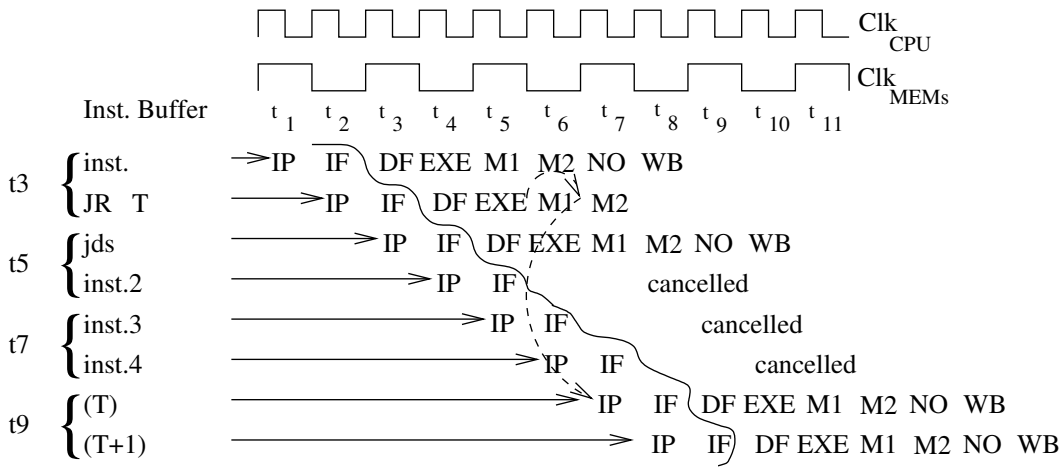

Figure 5.9: Target instruction latency induced by a *relative jump* instruction located in the IP stage: the target address, T, of *relative jump* instructions (JR) located in the IP stage are computed, or more precisely read from a specified register in the EXE stage (to avoid data dependency with *inst.*) and given to the instruction memory as quick as possible which is at t_7. As a result, the instruction within the *jump delay slot* (jds) is performed and the three next, *inst.2*, *inst.3* and *inst.4* are undesirable and then are discarded. The target instruction arrives at t_9.

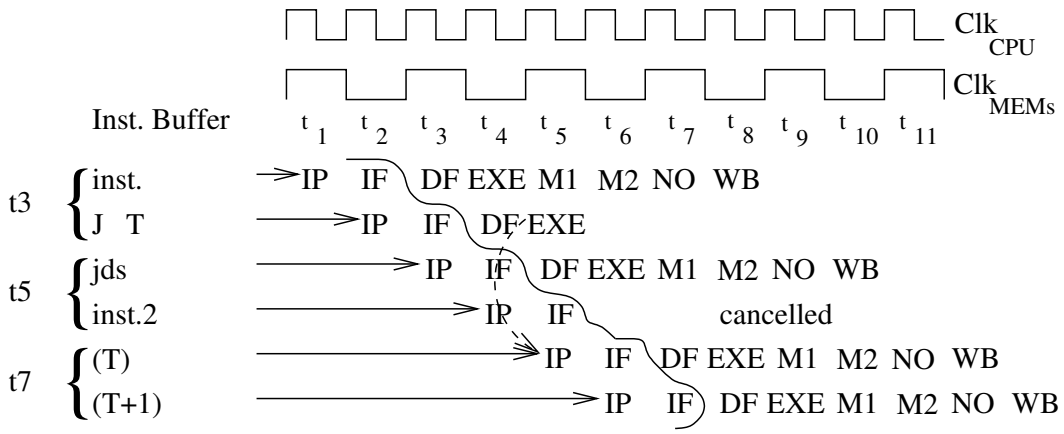

Figure 5.10: Target instruction latency induced by an *absolute jump* instruction located in the IP stage: the target address, T, of *absolute jump* instructions (J) located in the IP stage are computed, or more precisely read within the instruction in the DF stage and given to the instruction memory at t_5. As a result, the instruction within the *jump delay slot* (jds) is performed and the next, *inst.2* is undesirable and then is discarded. The target instruction arrives at t_7.

---

and CPI.

	JR	J
IF	jr	j
	jds	jds
	stall	stall
	stall	stall
IP	jr	j
	jds	jds
	stall	stall
	stall	(T)
	stall	(T+1)

(a)

	JR	J
IF	min.: 3 cc	min.: 3 cc
	max.: 4 cc	max.: 4 cc
IP	min.: 4 cc	min.: 2 cc
	max.: 5 cc	max.: 3 cc

(b)

Table 5.3: DSR-HP's Jump Control Flow (a) and CPI (b): the jump target is just established at the end of the EXE stage. For DSR-HD, the maximum CPI value may be increased by one (stall) like it is the case for taken branches (see tables 5.1 and 5.2).

5.3.3 Load Latency

The data dependencies due to *load* instructions can be divided into two groups of patterns of instruction sequences, each of which being made up of two or three cases[4]:

- the *load* instruction is decoded during the first half of the memory clock period—i.e., it was previously located in the IF stage—and is followed by either a memory instruction or by a non-memory instruction: in both cases, a *load delay slot* equal to two is mandatory to prevent a data dependency, as illustrated in figure 5.11.

  one can note that a *branch* does not cause a data dependence with the loaded data because this data is not read since the condition of the branch is predicted. Likewise, neither a J or JR jump instruction induces a data dependency because either the jump is *absolute*, which implies that none register is accessed, or the jump is *relative* but the register access is delayed until the EXE stage as detailled in subsection 5.3.2.

- the *load* instruction is decoded during the second half of the memory clock period, i.e., it was previously located in the IP stage: whenever a data dependency is detected with the third next instruction, a stall is mandatory in addition to the two *delay slots* to prevent a dysfunction. Three instruction sequences lead to data dependency:

  - a *load* instruction followed by a dependent *branch/jump* instruction (see figure 5.12).
  - a *load* instruction followed by a other dependent memory instruction (see figure 5.13 (b)).
  - a *load* instruction followed by a dependent *non-memory* instruction (see figure 5.13 (a)).

[4]In addition to the data dependencies describe in this paragraph, DSR-HD has the delays detailed in subsection 4.6

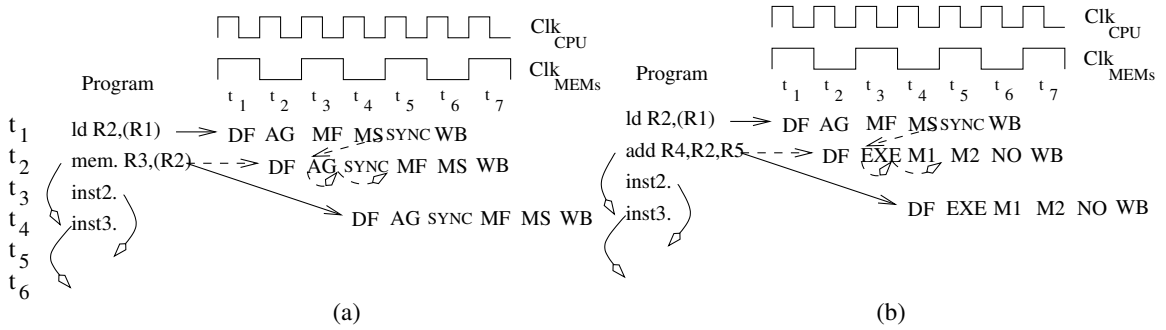

Figure 5.11: DSR's Load Delay Slots (IF stage): (a) Data dependency between a *load* and a next memory instruction. (b) Data dependency between a *load* and a next non-memory instruction.
The value contained in *R2*, *(R2)*, is not available until t_5 but in both cases, *R2* is required two cycles too early, at t_3 by the next instruction (*mem* and *add* in the AG and EXE stages, respectively). Two *load delay slots* allow to execute the dependent instructions two cycles later without stalling the pipeline.

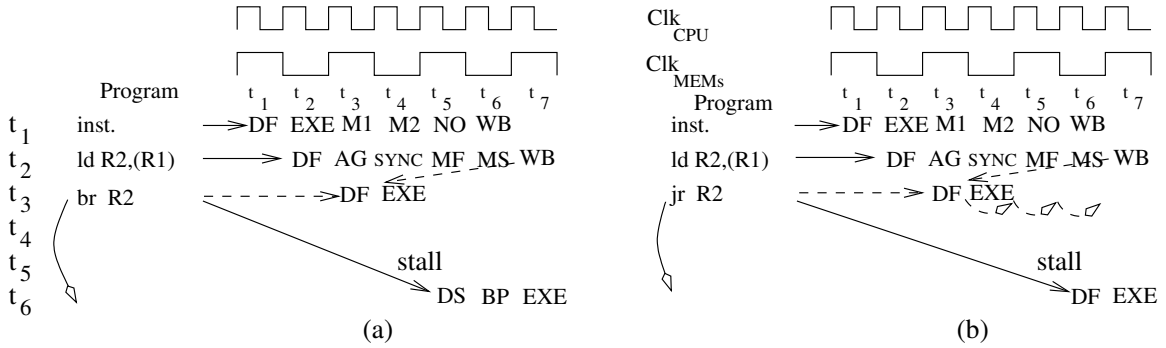

Figure 5.12: DSR's Load Delay Slots (IP stage) with Branch and Jump dependencies: (a) Data dependency between a *load* and a next *branch* instruction. (b) Data dependency between a *load* and a next *relative jump* instruction.
The value contained in *R2*, *(R2)* is not available until t_7 but in both cases, *R2* is required three cycles too early, at t_4 by the next instruction (*br* and *jr* in the EXE stage). Due to the dependency, both *branch* and *relative jump* fill none of the two *load delay slots*. So, they are located three instructions after the *load* in the code. Logic checks a data dependency with the *load* and stall the pipeline if any. In that condition, they passes from IF stage position to the IP stage position: the *branch* becomes predicted and the *relative jump* execution required a other stall (see table 5.3 (a) line IP).

Another kind of dependency between instructions exists, due to the dual port data L1 memory of DSR-HP, that is not based on operands. Indeed, a dependency may reside in writing or reading at the same memory location. These data hazards are rare and classified upon four types that depends on the order of consecutive write or read instructions. Consider two consecutive memory instructions, i and j where i is

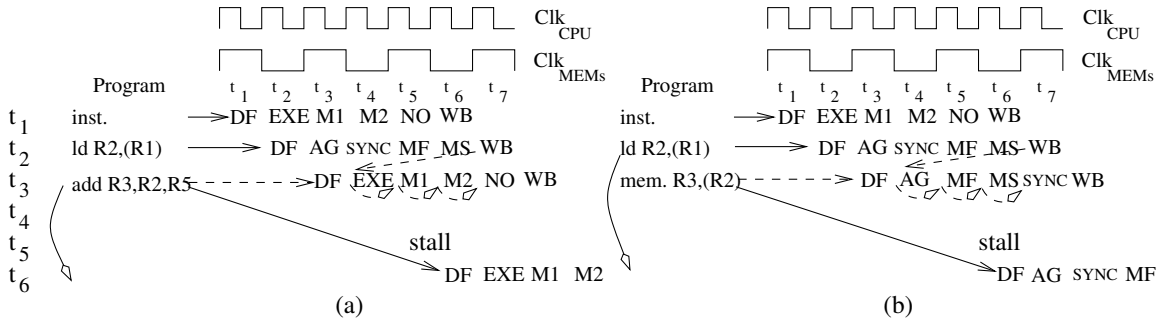

Figure 5.13: DSR's Load Delay Slots (IP stage) with Memory and Non-memory dependencies: (a) Data dependency between a *load* and a next non-memory instruction. (b) Data dependency between a *load* and a next memory instruction.
The value contained in *R2*, *(R2)* is not available until t_7 but in both cases, *R2* is required three cycles too early, at t_4 by the next instruction (*add* and *mem.* in the EXE and the AG stages, respectively). Due to the dependency, both *add* and *mem.* fill none of the two *load delay slots*. No extra stall is necessary.

decoded during the second half of the memory clock cycle, before j:

- Read After Write (RAW): j tries to read a data in the memory that is not updated by i yet. j reads an old value.

- Write After Write (WAW): i is the last write instruction that updates the memory. Hence, the memory does not hold the correct value.

- Write After Read (WAR): j writes at the wrong memory address when the register that contains the address has not been updated by i.

- Read After Read (RAR): though this case deals with operand dependency, it may arise still at a double memory access. A same register is finally updated by i instead of j.

Table 5.4 summarizes the previously mentioned memory reference dependency for DSR-HP with instruction sequence patterns where the first instruction involved is a *load*.

5.3.4 Store Latency

Table 5.5 summarizes the memory reference dependency describes in the above subsection but with instruction sequence patterns where the first instruction involved is a *store*.

	WAR	RAR
ld R2,(R1) **st R3,(R1)**	-	
ld R2,(R1) **st R3,(R4)**	-	
ld R2,(R1) **ld R2,(R4)**		-

Table 5.4: Memory Reference Hazards of Load Instructions: WAR dependency may not occur in the DSR-HP architecture because a *store* instruction is always buffered before accessing the memory: hence, leaving time to load the register *R2* cancels the hazard, even when the addresses *(R1)* and *(R4)* are equal. RAR hazard may not occur either because, during the MF stage, address comparison is made and in case of two consecutive *loads* that would like to update the same register, the value returned from the memory for the first of the two *loads* will not be taken into account.

	WAW	RAW
st R2,(R1) **st R3,(R1)** **st X,(Y)**	-	
st R2,(R1) **st R3,(R4)** **st X,(Y)**	-	
st R2,(R1) **ld R3,(R1)**		-
st R2,(R1) **ld R3,(R4)**		-

Table 5.5: Memory Reference Hazards of Store Instructions: neither WAW or RAW are possible in the DSR-HP architecture because time remains to both calculate (AG stage) and compare memory references (MF stage). Thus, in case of WAW, the older concerned *store* is removed of the write buffer, even when the addresses *(R1)* and *(R4)* are equal: only the pattern of figure 5.7 part (e) would have led lead to such a hazard where three consecutive *store* can induce a simultaneous data cache access and where the third *store* operands are not a concern. The issue of RAW dependency is solved by taking the data from the write buffer instead of the data cache, even when the addresses *(R1)* and *(R4)* are equal.

5.4 A RTL Implementation of DSR

5.4.1 A DSR-HP Simplified Version

A purged RTL version of DSR-HP has been designed in VHDL (see the full architecture specification of DSR-HP in section 5.2) that fits the requirements of a networking application in charge of routing IP packets on Internet edge routers.

Precisely, this version does perform integer operations—i.e., neither Floating Point Unit (FPU), multiply or divide operators are implemented. Furthermore, due to the fact that the data memory space did not exceed 8KB, no MMU was necessary, excepted a Fixed Map Translation (FMT) that just translates the program addresses into physical addresses according to the memory mapping before accessing the on-chip SRAM data memory (no caches). This RTL design does not use a BTIC because few *branch* and *jump* instructions occur in the targeted application.

The previously mentioned network application led to an implementation of a five-stage RISC processor, called X5 with the same above features.

5.4.2 Technology and Tools

The CPU core and its instruction and data on-chip memories—two dual port 2048x32 High-speed SRAM memories from Virage Logic with a *column mux* equals to 16 [66]—were synthesized with the Synopsys's *Design Compiler* under timing constraints [63] and a zerowireload model. The targeted technology was the TSMC's 0.13μm CMOS and low threshold voltage process, called *CL013LV* [64].

The generated VHDL format netlist, which take into account cell delay was used with the Synopsys's VHDL System Simulator (VSS) to achieve simulation at the gate level, and then (partially) check the functionality of the design.

The verification of the design functionality was made with a *in-house* instruction sequence generator and a software behavioral model of the processor.

The Virage Logic memory IPs were generated with the *ts13e1p11hssb05* compiler which is configured to build High-performance Synchronous High Speed RAM using the TSMC CL013LV-OD-FSG CMOS process.

5.4.3 DSR's Features

The DSR-HP processor core described above uses 67,000 gates that represents a silicon area of about 240161μm^2 (nand area = 5.0922μm^2). The bare core runs at 625MHz (1.6 ns) under the worst operating conditions (1.08V, 125^{0}C): this frequency does not take into account back-end parameters (layout, wires, place&route, ...) and depends on the quality of the mask. For instance, if we assume a frequency degradation of 20%, the frequency will be equal to 500MHz.

The critical path is located in the execution unit—from the ALU's adder to the forwarding multiplexer that returns the result back to the ALU itself—and the circuit uses scan test flip-flops. The mentioned multiplexer takes about 60% of the clock period and is equal to 0.72ns (i.e., 22 FO4s) after having taken into account a 20% of back-end delays.

The X5 processor has been fully tested, and the chip (back from the foundry) uses 30,000 gates and runs at 350MHz with a critical path that is still shared by the main adder and the forwarding multiplexer (roughly equal to $\frac{2}{3}$ of DSR and that takes 15 FO4s).

5.5 Conclusion

The DSR architecture allows to cut the critical path of five-stage processor that needs a single cycle to jump to a targeted instruction: this is due to the fact that the branch condition is checked in the ALU stage and no more in the ID stage, and then the ALU-ID forwarding path is cut. Also, one can imagine to move some logic from a stage to a other like other vendors did in order to better balance the logic on all the stages.

Consequently, this chapter shows up that the implementation of a pipeline made up of eight stages like DSR-HP should reach the same frequency than the last MIPS32 24K, ARM11 and ARC700 processors if the same back-end were used: indeed, the X5's clock frequency (see subsection 5.4.3) is roughly the same than its counterparts in the industry (see table 2.1), and then 550MHz should be reach for a deeper eight stage processor.

The added value of DSR will come of the fact that this frequency is not (or less) deteriorate when the L1 memory access time is taken into account.

Finally, DSR uses twice slower L1 memories than those proposed by the processor vendor pre-cited, what contributes to reduce power consumption and to save silicon area by using slower and hence smaller L1 memories.

Chapter 6

Software Support Tools

6.1 Introduction

This chapter describes the software environment in which the results presented in the next chapter were obtained. It regroups a full cross compiler tool chain, a RISC processor simulator and a profiler, and a test environment based on reference embedded programs. The timing metric used in this document is also defined.

Initially, a software that gives features on the area, timing and power consumption of SRAMs according to their organization and technology node is described and compared to Virage Logic's counterparts.

6.2 The Timing Metric

The goal is to use a timing metric that remains constant over various process technologies, voltages and temperatures. Moreover, this metric has to characterize static and dynamic CMOS gates that dominate digital designs. An inverter driving four identical copies of itself was chosen as the elementary gate delay [23]: due to its capacitive fanout of four, this delay was called a *fanout-of-four inverter delay*, know as FO4. For a given technology, the FO4 delay is close from one company to another. However, sophisticated in-house technologies such as from Intel or IBM can deliver faster FO4 due to expensive efforts [21].

Historically, the FO4 delay was set to $500.L_{gate}$ ps under worst case operating conditions (low Vdd, high temperature). As of the 0.13μm technology node, the physical gate length, L_{gate} became shorter than the drawn feature size because the width of the gate material specified by the layout engineer (mask setting) is shortened by

the etching process and the lateral *source-drain* diffusion. Furthermore, the International Technology Roadmap for Semiconductors (ITRS) indicates a physical gate length trend to be roughly half the DRAM $\frac{1}{2}$ pitch which is identified as the feature size. As a consequence, the previous expression was extrapolated to $250.L_{node}$ where $L_{node} \simeq \frac{1}{2}L_{gate}$.

Year	**2002**	**2004**	**2007**	**2010**	**2013**	**2016**
Technology node (nm)	130	90	65	45	32	22
wc FO4 delay (ps)	32.5	22.25	16.25	11.25	8	5.5

Table 6.1: FO4 delay as a function of CMOS technology node: according to [60], the feature size decreases of a theoretical scaling factor 0.7 at each high performance process node, as it happens for the power supply and the clock frequency in wc operating conditions. In the FO4 delay formula,L_{gate} is expressed in microns.

Thus, any delay can be normalized with FO4. he timing comparison is reported in term of FO4 to be independent of the technology node.

6.3 CACTI, a Cache Access and Cycle Time model

CACTI [44] is a recognized popular tool used by computer architects for determining the optimal cache configuration for a given set of parameters like the cache size, the block size, the associativity, the number of read and write ports and their size, and the technology node. Usually, it helps to explore cache configurations for area, power and access times.

The last 3.2 version of this tool was essential in this work to show the gap and the improvement of the DSR architecture in comparison to existing ones. Indeed, it provides information the access time on various configuration of on-chip memories for many technological nodes on a typical process. In fact, because as of the 0.18μm node, the bitline and sense amplifier delay of a SRAM do not scale as much as logic, the whole on-chip SRAMs do not scale neither [5][12][39]. That is the reason why, first, CACTI was used to determine cache timing access for the 0.18μm node, and then, below this technological node, the 0.7 scaling factor of ITRS (resumed by CACTI) was replaced by a scaling factor that takes into account the bitline and sense amplifier overhead: the 10% delay overhead of [1] was resumed in this work and the scaling factor became even to 0.77. Furthermore, this new factor was not applied linearly with the feature size as it is the case with CACTI: indeed, the new timing access

factor was applied gradually as of the 0.18μm technological node.

Whereas the gate delay decreases by 50% each two generations, the on-chip cache access time decreases by 40.7% and even 42% in [12].

In order to validate the access time results of CACTI3.2, they were compared to SRAM memory IPs of Virage Logic [66] based on the TSMC 0.13μm High-performance, low threshold voltage process, called CL013LV-OD-FSG, and generated with the ts13e1-p11hssb05 compiler. For consistency reasons with CACTI, the results obtained under a typical Process Voltage Temperature (PVT) (1.2V - 25^{o}C) are taken into account to establish the timing error induced by the software model (see figure 6.1): it appears that the average access time error between the Virage Logic' IPs and the CACTI's models is quite small (about 2 FO4s) compared to an average access time of 30.4 FO4s. More precisely, the average error is about 6.7% and varies between 2% in the best case and 16.4% in the worst case.

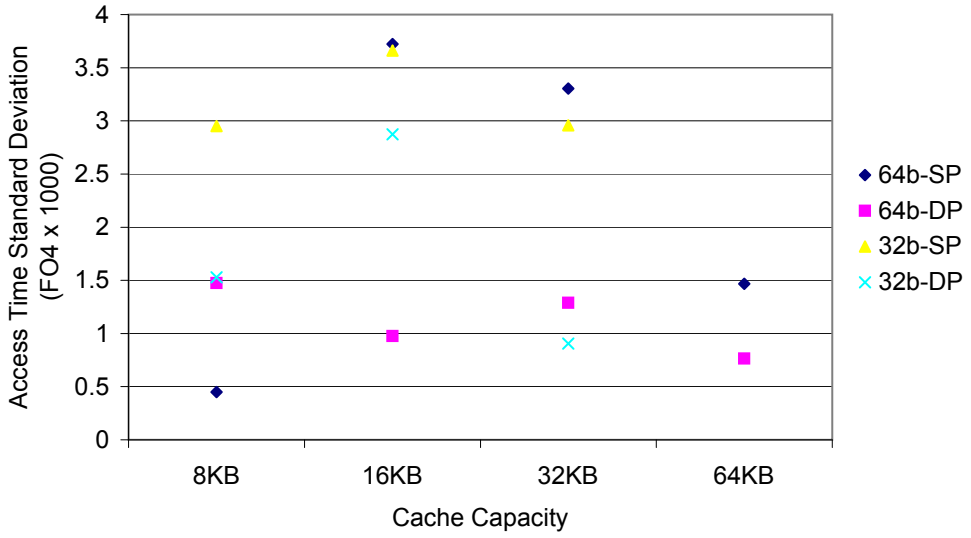

Figure 6.1: Access Time Error between Virage Logic's SRAMs and CACTI's one-way Memory Models: these results are for a 0.13μm technology node in a typical process (CL013LVOD for Virage Logic) where wire delay are added to the CACTI3.2 results as 10% of the clock cycle. All the comparisons are made on direct mapped caches with the same configuration (block size, etc.). The 8KB and 16KB 64-bit single port direct mapped memories are, in term of errors, the best and worst cases, respectively. The standard deviation is just the access time difference between two equivalent caches.

However, for industrial reasons, worst-case operating conditions were considered to established processor performance in chapter 7. One noticed a factor 0.7 to pass

from a Virage Logic's access time under a typical PVT (1.2V - 25^{o}C) to one based on a worst-case PVT (1.08V - 125^{o}C) access time. Consequently, the access time results provided by CACTI were consciously deteriorated by a factor of 0.7 to fit with industrial conditions.

Finally, a worst case access time factor that allows to pass from a High-speed (HS) to a High-density (HD) SRAM, for a given technology node, was determined: it is the average of fourteen pairs of identical memories based on the CL013LV-OD process and was identified as equal to 0.67. This factor helps to establish the clock period and performance of processors (L1 caches included) dedicated to use the smallest amount of silicon area than possible (see section 7.4).

6.4 The Cross Compiler

In many system contexts, notably embedded ones, a cross tool chain may be necessary to compile a high level user code dedicated to run on a processor different from the host machine.
The cross tool chain mentioned in this paper is based on GCC, a widespread free C compiler [57]. It is made up of six major components: the cross compiler itself (e.g., GCC version 2.95.2), an assembler (GAS) and a linker (GLD) based on various utilities (e.g., Binutils version 2.13), a library (e.g., newlib version 1.12.0) and a debugger (e.g., GDB version 6.0).

A generic flow was identified and implemented to settle a customized cross tool chain according to a determined processor architecture like DSR [35]: it allows to remove, add or modify instructions from a given GCC ISA and to manage *delay slots* according to the processor architecture. In the case of this study, GCC targeted for the MIPS 1 ISA was tailored to match with the DSR pipeline.

6.5 The VMIPS Tool

VMIPS is a virtual machine simulator based around a MIPS R3000 RISC CPU core [56] and its MIPS 1 instruction set that executes integer programs. It is an open-source project written in C++ and which is distributed under the GNU General Public License. VMIPS, being a virtual machine simulator, does not require any special

hardware. It has been tested under Intel-based PCs running FreeBSD and Linux, and a patch has been developed for compatibility with CompaQ Tru64 Unix on 64-bit Alpha hardware.

VMIPS comes up with a full set of MIPS-targeted cross-compilation tools, and the build process assumes their existence on the host system.

VMIPS can be easily extended to include more virtual devices, such as frame buffers, disk drives, etc. VMIPS is also designed with debugging and testing in mind, offering an interface to the GNU debugger GDB by which programs can be debugged while they run on the simulator.

VMIPS was settled in combination with the customized cross compiler mentioned in the previous section. Furthermore, in order to be able to execute any C program, a Virtual Operating System (VOS) has been developed which allows to performed input/output functions like *printf*, for instance, even if it is not relevant for embedded applications.

A profiler has been also attached to VMIPS that helps to establish statistics on the program that runs on a R3000 24K MIPS or DSR processors: e.g., it gives information on the number of read and write access to the memory, the number of hits, the number of *delay slots* that are filled by a useful instruction, the number of *branches* and *jumps*, the overall number of executed instructions, and so on. Moreover, the multiplication and division latencies were set to 5 and 35 clock cycles, respectively, as suggested in [48].

All the mentioned enhancement brought to VMIPS are detailed in [2].

6.6 Test and Verification

Recently, it has become popular to put together collections of benchmarks to try to measure the performance of the processors with a variety of applications. Of course, such suites are only as good as the constituent individual benchmarks. Nonetheless, a key advantage of such suites is that the weakness of one is lessened by the presence of the other benchmarks. Benchmarks for embedded systems are in far more nascent state than those for either desktop or server environments. They were used in this work to validate the VOS, the profiler, the cross compiler and the complete flow.

For those embedded applications that can be characterized well by kernel perfor-

mance, the best-standardized set of benchmarks appears to be the EDN Embedded Microprocessor Benchmark Consortium (or EEMBC pronounced embassy) [55]. The EEMBC benchmarks reflect real-world applications and the demands that embedded systems encounter in these environments. The result is a collection of "algorithms" and "applications" organized into benchmark suites targeting telecommunications, networking, automotive/industrial, consumer, and office equipment products. Although many embedded applications are sensitive to the performance of small kernels, remember that often the overall performance of the entire application is also critical. Thus, for many applications, the EEMBC benchmarks can be used to partially access performance. A few EEMBC benchmarks involve floating-point calculations for time calculations and displays, which can be avoided by changing one option (FLOAT_SUPPORT) in the EEMBC environment since the MIPS R3000 does not support floating-points.

All of the benchmarks used in this work (42 programs listed in appendix E) were compiled under the same conditions, i.e., with the `O3` and `funroll-loops` setting options of GCC which try to improve the program time execution. Due to "relocation truncated" errors, the `-G 0` option was applied in all cases as specified in [49]. The libraries were generated in the same way.

Finally, figure 6.2 illustrates the full verification flow made up of the customizable cross compiler tool chain, the simulator, the profiler and the EEMBC benchmarks that act as a verification tool.

6.7 Conclusion

The timing metric FO4 for deep-submicron technologies has been defined in this chapter as 250 times the technological node length.

Moreover, because the standard deviation of the CACTI and Virage Logic SRAM access time is quite small—2 FO4s over an average access time of 30.4 FO4s—CACTI was used in this thesis work to generate various on-chip memories in a 130nm technology, then a new access time factor equal to 0.77 is gradually applied to subsequent technological node. Indeed, the access time error (or standard deviation) between a CACTI or Virage Logic memory is about 6.7% and fluctuates between 2 and 16.4%.

Finally, a full customizable tool set including a cross tool chain, a simulator, a

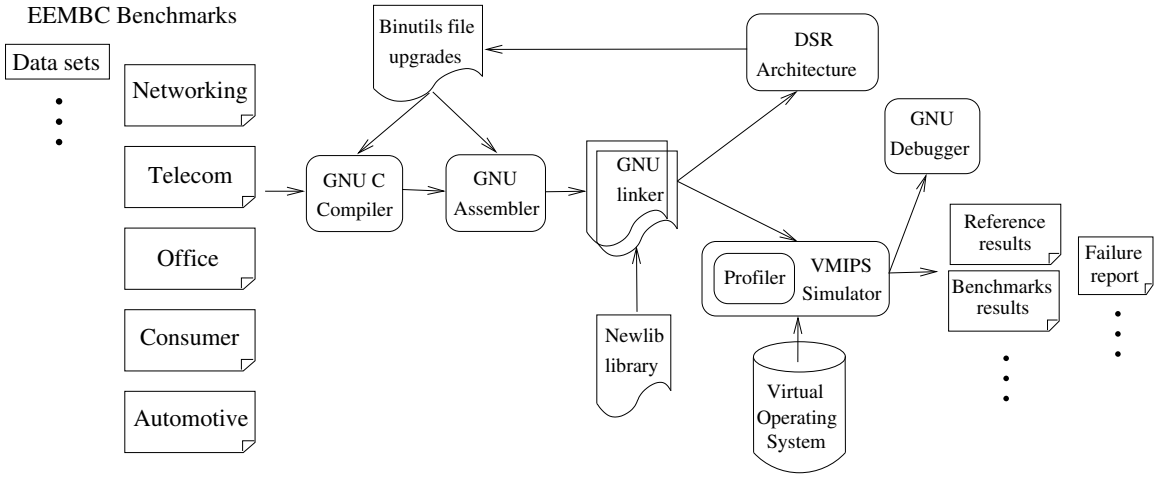

Figure 6.2: DSR Software Environment: the starting point of this flow is the chosen DSR architecture that allows to set the cross tool chain and the simulator. Afterwards, user's programs can be compiled and/or assembled in order to be executed on the simulator or the DSR processor itself. The *reference results* that come with the EEMBC benchmarks are compared to the simulator outputs to establish *failure reports*: then, cross tool chain or simulator misfunctions can be highlighted. Furthermore, performance results are given by the embedded profiler.

profiler and a test environment was implemented.

Chapter 7

Results

7.1 Introduction

The results are achieved considering architectures based on a same ISA (MIPS 1) and using a same compiler (see chapter 6) in order to compare the performance and efficiency of the MIPS R3000, MIPS24k and DSR-HD architectures themselves. This is feasible and relevant because the R3000 and the 24k are based on the MIPS 1 ISA and DSR was voluntarily elaborated under this same ISA.
Moreover, in order to establish accurate results, the data and instruction memory sizes were determined for each applications.
Also, we can note that only Direct Mapped (DM) memories were used in this work.
The first part of this chapter deals with the tendency of the wire delays, primary memory access times and bare processors frequencies, according to future technology nodes.

Also, this chapter highlights the performance, which is a mix between the frequency and the number of instructions required to execute a program (also known as Million of Instructions Per Second or MIPS).

Finally, the last part focuses on the main relevant metric of a processor, its efficiency, which is the ratio between the performance and the surface required: it shows how relevant DSR-HD is, compared to the MIPS R3000 and the MIPS24k.

7.2 Wire Delay Trend

A given process has several layers of copper interconnect, where upper layers are wider and taller than lower ones. These layers may be grouped into three categories

of interconnections:

- The lowest metal layer which has the finest pitch and hence the highest resistance: it is used to transport information between nearby gates and inside gates themselves. Wires in this layer are called local wires and may have a pitch of about 5λs[1].
- Middle metal layers that have wider pitch than the previous category and that route information within functional units. Wires in this layer are called semi-global wires and may have a pitch of about 8λs.
- The top layers that have the widest pitch and hence the lowest resistance: they carry global routes, clock power and ground. Wires in this layer are called global wires and may have a pitch of about 16λs.

Furthermore, one can distinguish two types of wires among these three layers: one type used to connect gates and blocks, called *local* that will scale with the logic gates and the second one, called *global* that will not scale so much because they cross significant part of the die. Each can have a short or long length. As a consequence, one focuses on semi-global scaled-length wires because our concern is the CPU and L1 cache communication [12][42]. Figure 7.1 shows the trend of the CPU-L1 cache wire delay according to technology nodes and based on two projections of wire technology scaling: one considered optimistic (or aggressive) based on small resistance degradation, low-k dielectrics and the other considered pessimistic (or conservative) including scattering effects. The consensus is that the future wire characteristics will fall inside this range [21]. These values are extracted from [24].

Due to a quadratic relation with wire length, long wires have an irksome long delay. Such long wires can be divided into small segments by the use of gain stages—called repeaters—in between them to obtain length-squared delay relation. However, these device occupy a large silicon area when used in buses [21].

Furthermore, a side effect of this thesis work is to reduce the performance gap between embedded microprocessors designed by large teams and those created by small companies. The goal is to improve their ASIC design methodology by combining their automated Computer Aided Design (CAD) flows with an architecture that avoids the delay penalty of wire and SRAM scaling: indeed, a small structure cannot afford to manually optimize a layout for cost and time-to-market reasons and their CAD tools poorly estimate critical wire path delays. As a consequence, repeaters or other *superwires* are not ideal solutions for the embedded world, even when they are delay-power optimized.

[1] λ is half of the drawn gate length and it is a way to describe pitches in a technology-independent manner.

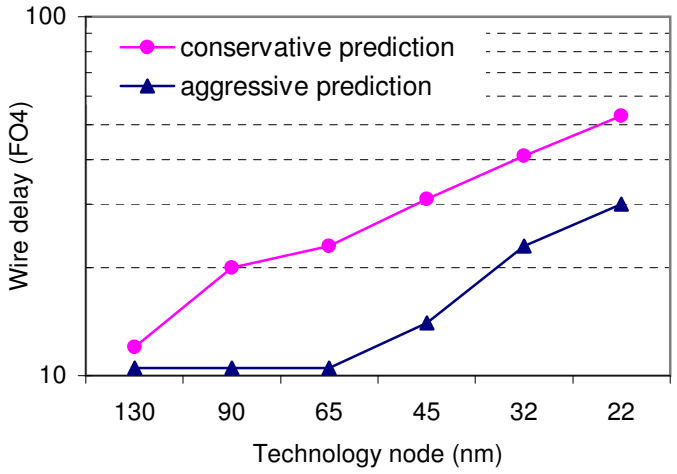

Figure 7.1: Wire delay scaling spanning 50K gates: the semi-global (unrepeated) scaled-length wires, used to connect a CPU core and its closest on-chip memory, have a delay that grows by a factor of about 2.9x to 4.3x over six generations. The upper and lower curves indicate conservative (pessimistic) and aggressive (optimistic) scaling trends, respectively.

7.3 On-chip Memory Access Time Trend

The MIPS R3000, the MIPS32 24K and DSR-HP require single port 32-bit, single port 64-bit and dual port 32-bit SRAMs as L1 memory, respectively (DSR-HD uses single port 32-bit and 64-bit SRAMs). Most embedded applications require a small amount of memory: as a consequence, this study is made on on-chip direct-mapped SRAM memories, rangeing from 8 to 64KB. the results have shown that single port 32-bit and 64-bit HS SRAMs have the best (smallest) and the worst (longest) access time on a CL013LV-OD process. So, only these two types of memories were reported in figure 7.2 to show how memories scale with the technology nodes. As discussed in detail in section 7.4, we may already have a feeling for the fact that the memory access time has a negative effect on the global clock period, a phenomenon that is expected to worsen with future technology generations.

In any case, from 8 to 64KB, the memory access time scales slower than the logic speed: a loss of about 55% is seen between the 130 and 22nm nodes. This is mainly due to the interconnections within the memory itself that scale more slowly than the transistors (this is described in more detail in section 6.3).

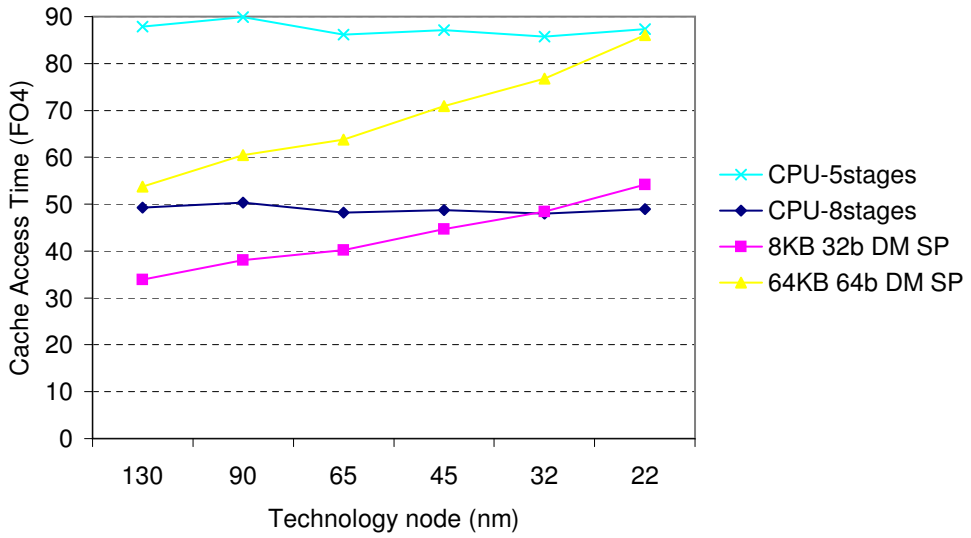

Figure 7.2: Cache Access Time vs. CPU Clock Period: the access time of any relevant on-chip memory configurations is in between the drawn single port memory curves, the 8KB 32-bit being the best case in term of access time and the 64KB 32-bit the worst case. Even a 8KB memory will deteriorate the clock period as detailed in the section 7.4. The CPU frequencies are extracted from table 2.1.

7.4 Scalar Embedded Processor Clock Frequency Trend

We have seen in chapter 2 (table 2.1) that a five-stage and an eight-stage synthesizable cores could run up to 350 and 625MHz ($T_{barecpu}$) on a CL013LV-OD process, respectively. Coupling this and the ITRS speed predictions to the on-chip memory access time ($T_{mem.access}$) and wire delay trends (τ_{wire}) developed in the previous sections, the clock period (T_{clk}) of the whole system (core and L1 memories) is established with the equation 7.1, 7.2 or 7.3 for both High-performance and High-density configurations as shown in figures 7.3 and 7.6, respectively.

Furthermore, the experience have shown that the critical path balances between the ALU's adder followed by the global multiplexer that forwards the calculated result back to the entrance of the ALU stage and the data read in the data L1 memory that also goes to the entrance of the execution stage through the same multiplexer [12][50]. The first mentioned possible critical path determines the maximum clock period values as reported in table 2.1 whereas the second path becomes critical below a certain on-chip memory size. Consequently, this multiplexer, often the biggest in the core is also accounted for the clock period prediction (as τ_{fwdmux}) and obviously scales as fast as logic (see section 5.4.3).

In addition to what preceeds, parameters like latch, skew and jitter overheads (grouped under the name $\phi_{overhead}$) degrade the clock period of a processor [25]. Part of this whole delay overhead does not scale with technology and was set to 1.8 FO4s. This value becomes 2 FO4s in this thesis work because our FO4 is defined as $250.L_{node}$ instead of $360.L_{node}$ (see section 6.2) as in [12][25].

Equation 7.1 models the entire clock period of the MIPS R3000 used to create the graphs to come. Similarly, equation 7.2 represents the model of the MIPS24k clock period where the forwarding multiplexer has been removed from the critical path because the data loaded from the memory is available one cycle later. Finally, equation 7.3 corresponds to the DSR-HD period where the primary memories have one more CPU cycle to return their data to the core.

$$T_{clk} = max(T_{barecpu}, max(T_{inst.mem.access}, T_{d.mem.access} + \tau_{fwdmux}) + \tau_{wire} + \phi_{overhead}) \tag{7.1}$$

$$T_{clk} = max(T_{barecpu}, max(T_{inst.mem.access}, T_{d.mem.access}) + \tau_{wire} + \phi_{overhead}) \tag{7.2}$$

$$T_{clk} = max(T_{barecpu}, \frac{max(T_{inst.mem.access}, T_{d.mem.access} + \tau_{fwdmux}) + \tau_{wire} + \phi_{overhead}}{2}) \tag{7.3}$$

7.4.1 Speed Optimization: HP Configuration

DSR-HP involves dual ports as primary memories as described in section 4.3. Furthermore, HS IP memories are considered (opposite to their HD IP counterparts) because this section deals with High-performance rather than with area. Thus, the results will show how far the processors can go, in terms of frequency.

In a 45nm technology node, a small memory such as 8KB is part of the the MIPS R3000 critical path whereas the speed already saturates in the 90nm node when 64KB memories are used.

The saturation comes earlier for the MIPS32 24K than for the R3000, i.e., at the 90nm technology node with 8KB memories whereas DSR-HP stops following the bare 8-stage clock period at the 45nm node. Moreover, DSR-HP has a better behavior than

the two other architectures with a 64KB: indeed, it saturates later, i.e., at the 90nm node.

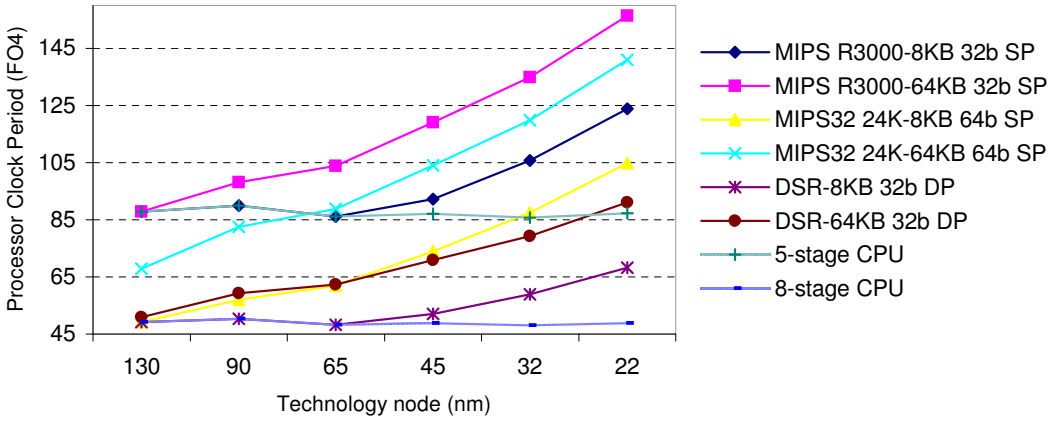

Figure 7.3: Embedded Processor Clock Period Scaling: three processor clock period scalings are reported in this graph including direct mapped primary memory access time, forwarding multiplexer, conservative wires and clock overhead. We can see that the bare CPU is not the critical path for long: even for the five-stage processor, a (small) 8KB data memory will be the critical path as of the 45nm technology node whereas the same path already limits the frequency at the 90nm node with a 64KB memory. The MIPS24K clock period is smallest but has roughly the same behavior as the R3000 one whereas DSR-HP is less sensitive to technology scaling and limits its effect for later technology nodes.

The clock period gap between both a five-stage processor and the MIPS32 24K and their corresponding bare CPU speed increases quickly as technology scales to reach a difference of 69 and 92 FO4s at the 22nm node in the 64KB configuration, respectively (see figure 7.4): large memories have an important negative influence on the MIPS R3000 and MIPS32 24K clock frequencies. This is due to the poor frequency scaling of on-chip memories compared to logic and the increasing importance of the wire delay. This effect is reduced for DSR-HP with only a gap of 42 FO4s, a 39.1% improvement.

At the 22nm node, even for small memories, DSR-HP suffers the least of the data that return from the data memory: e.g., in case of 8KB memories, the period saturation is about 37, 58 and 19 FO4s respectively to the R3000, MIPS24k and DSR-HP.

Measures show that the use of an aggressive wire delay model does not reduce the advantage of DSR. Indeed, at the 22nm technology node, DSR-HP still saves between 33-53% and 55-76% of FO4s compared to the R3000 and the MIPS24K in the 64KB configuration (see figure 7.5 and appendix A for a full graph using aggressive wire prediction). This due to the fact that, as of the 45nm node, even the optimistic wire

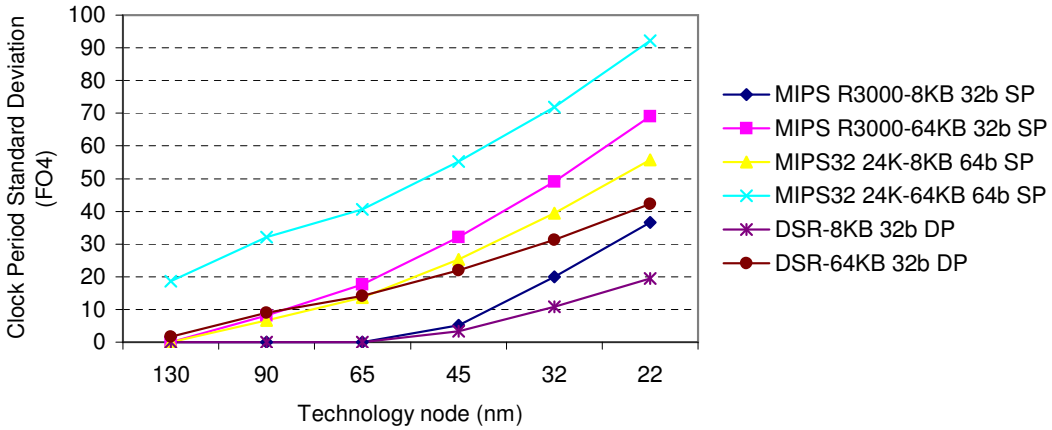

Figure 7.4: Embedded Processor Clock Period Standard Deviation: at each node and for any memory configuration, DSR-HP has the smallest clock frequency difference with it corresponding bare CPU frequency and the coefficient of degradation (modelled by the curve slopes) is the smallest.

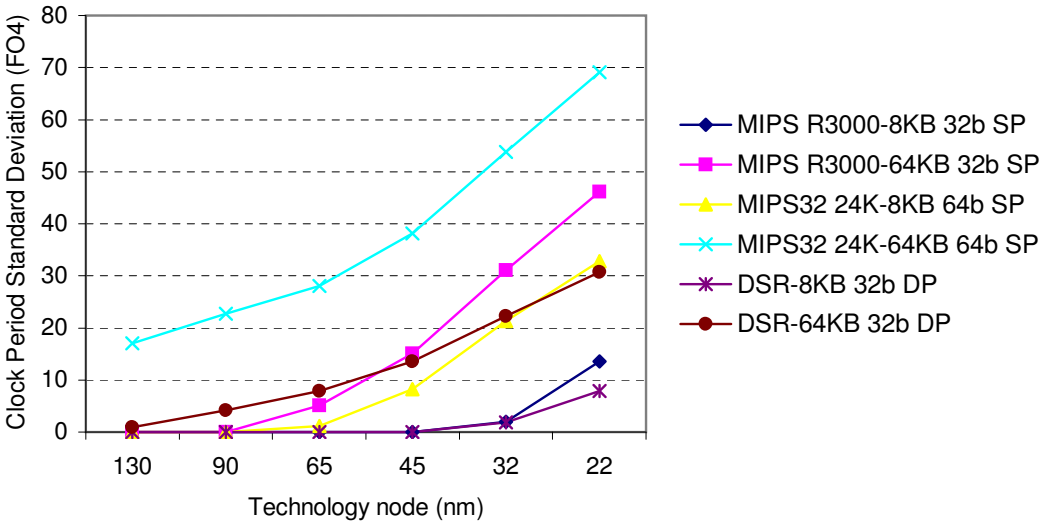

Figure 7.5: Aggressive HP Embedded Processor Clock Period Standard Deviation: the negative effect of the poor technology scaling of memories and wires is just delayed compared to conservative wire prediction: indeed, the data memory access becomes critical two technology nodes earlier than for the conservative wire prediction (22-32 vs 45-65 nodes).

prediction follows a curve with almost the same slope than the pessimistic wire delay prediction (see figure 7.1).

7.4.2 Area Optimizaton: HD Configuration

This paragraph presents the frequencies that the studied architectures can reach when the silicon area is the main criterion. As a consequence, HD IP memories are used because they involve a smallest amount of silicon.

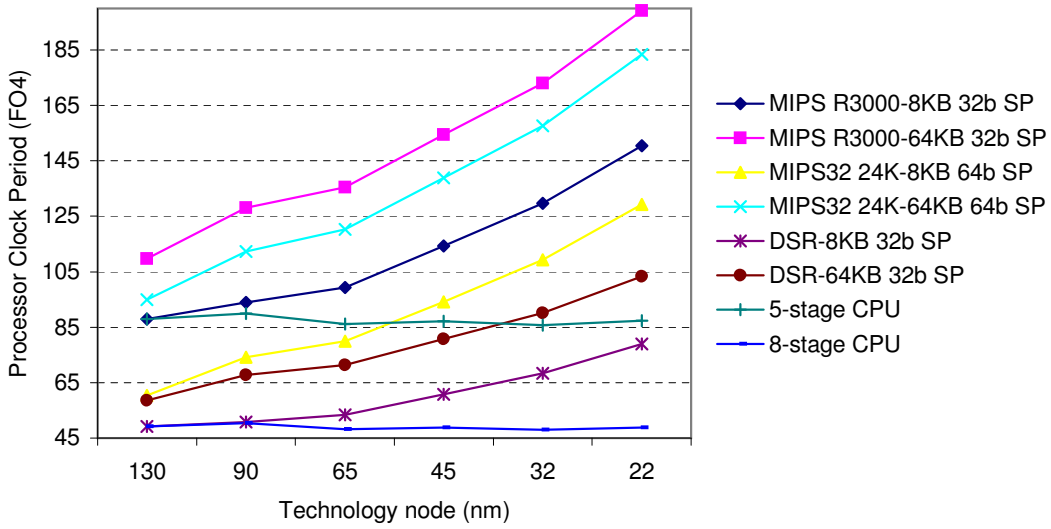

Figure 7.6: HD Embedded Processor Clock Period Scaling: the three processors always take into account direct mapped primary memory, forwarding multiplexer, conservative wires and clock overhead. The bare CPU is no longer the critical path as of the 65nm node for the five-stage processor with a (small) 8KB data memory, whereas the same path already limits the frequency at the 130nm node with a 64KB memory. The DSR-HD is less sensitive to technology scaling and limits its effect for later technology nodes. In any case, DSR-HD proposes the fastest clock frequency of complete micro-processors.

Except for the R3000 at the 130nm node and DSR-HD at the 130 and 90nm nodes used with a 8KB memory, the clock frequencies of all other configurations suffer of a slower memory and DSR-HD proposes the best behavior under technology scaling (see figure 7.7).

Compared to the HP speed graph 7.3, all the curves are vertically translated inducing a greater number of FO4s per clock period and hence longer periods (see figure 7.6). However, the deterioration of the clock period in terms of FO4s is restrained for DSR-HD. Indeed, the increase of the period reaches about 28% for the R3000, 30% for the MIPS24K and only 13% for DSR-HD (22nm node) in the 64KB configuration.

If one alleviates the wire effect with an optimistic wire model, the tendency stays roughly the same: the benefit of the DSR-HD solution occurs two technology nodes later, showing the increasing limitations of the current designs as technology improves.

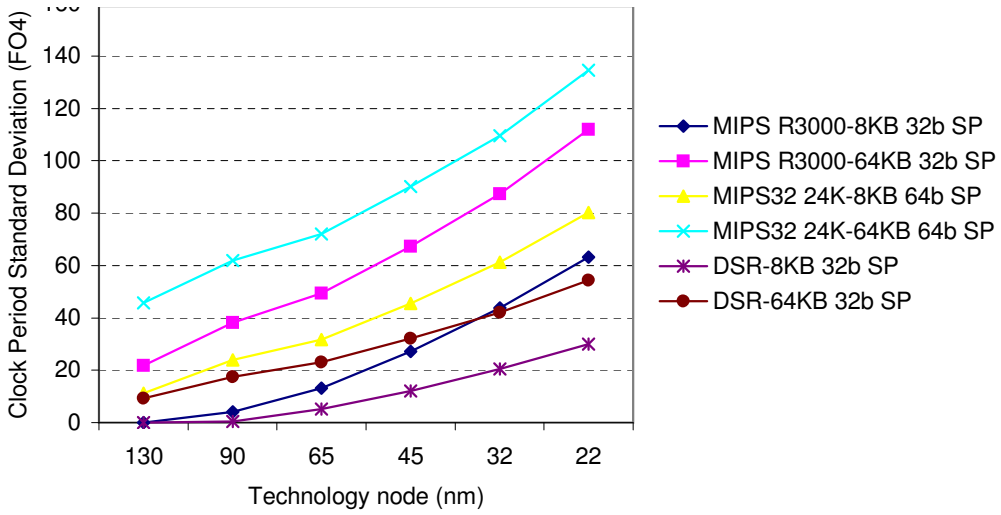

Figure 7.7: HD Embedded Processor Clock Period Standard Deviation: DSR-HD appears to be even earlier the best solution in terms of clock frequency with the generation to come because it counts the smallest number of FO4s between the effective and ideal clock period. In any case, DSR-HD is the architecture the least influenced by the memory speed.

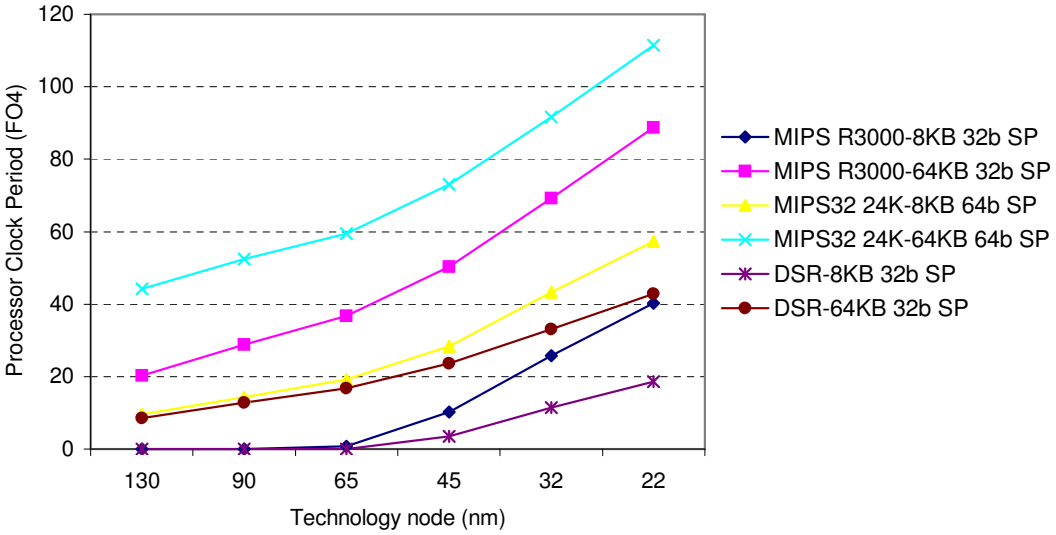

Figure 7.8: Aggressive HD Embedded Processor Clock Period Standard Deviation: Similarly to the HP configuration, the negative effect of the poor technology scaling of memories and wires is just delayed compared to conservative wire prediction: indeed, the data memory access becomes critical two technology nodes earlier than for the conservative wire prediction (45 vs. 90 node).

7.4.3 Conclusion

The number of FO4s that separates the R3000 and the MIPS24K speed to their ideal clock frequencies dramatically increases with the use of HD and then slow memories (see figure 7.7): e.g., for the 64KB configuration, the gap is equal to 112 FO4s for the R3000 and 128 FO4s for the MIPS24k whereas DSR-HD shows a relatively small difference of 48 FO4s. In case of 8KB memories, the differences become 63, 80 and 30 FO4s, respectively. So, DSR-HD alleviates even more the negative effect of technology scaling in the HD configuration than in the HP configuration: this is due to the use of HD memories that are slower than their HS counterparts and that have a straightforward negative effect on the R3000 and MIPS24K clock frequency (see paragraph 2.8). Moreover, DSR-HD no longer uses dual port memories—as it is the case for DSR-HP—but single port ones that are faster.

Furthermore, the delay of wires remains an issue—independently of the wire model—even if an optimistic algorithm is used as shown in figure 7.8.

To summarize, the slower (or larger) the memories, the greater benefit DSR, compared to the other two architectures in terms of frequency: the core and primary memory speeds are more homogeneous and allow to process the instructions under the smallest clock period, as defined in equation 7.1.

7.5 Scalar Embedded Processor Performance

The previous section shows that DSR can ensure the fastest clock frequency compared to the a five-stage processor and the MIPS32 24K. However, it does not mean that its performance—in terms of Million Instructions Per Second (MIPS)—is better because of its deeper pipeline that introduces cycle delays on branches and memory instructions compared to a five-stage processor, as detailed in section 5.3.

In this section and the next, we establish the performance of three processor architectures (the MIPS R3000, MIPS32 24K and DSR-HD) running 42 EEMBC integer benchmarks (described in section 6.6): indeed, we will not focus any longer on the DSR-HP architecture[2], mainly because the reduced surface of DSR-HD is more important than the gain on the clock frequency offered by DSR-HP with faster memories. By convenience, only the best, worst and typical case results obtained by DSR are exposed, the others being placed in appendixes I and J.

[2]DSR-HP performance results on EEMBC benchmarks are reported in appendixes G and H.

All the processors studied in this work executed the same embedded applications compiled with the same compiler with the identical options (see section 6.6). Furthermore, all of them are based on the same MIPS 1 ISA which implies that a single program contains the same number of effective instructions for all the processors. As a result, the performance of the raw processors and their memory subsystems themselves are compared. In addition, we consider that all the required memory is available in the primary memories due to its small size, and hence neither a multi-level hierarchy of caches is necessary, nor caches: i.e., simple SRAMs are used as primary on-chip memories for both instruction and data memories, where memory misses are prevented and Fixed Map translation (FMT) is sufficient. Finally, for consistency reasons, we consider for DSR the same maximum clock frequency as the MIPS32 24K: indeed, both are made up of the same number of stages and we postulate that the logic within DSR could be as balanced as that produced by the experienced MIPS's engineers to reach the 625MHz of their last MIPS processor.

Table B.1, in appendix B, gives the CPIs for each EEMBC integer benchmark on all the studied processors where the MIPS R3000 and the MIPS32 24K represent the five-stage and the eight-stage synthesizable processor world, respectively. These values were obtained from the VMIPS simulator (see subsection 6.5). The MIPS24k CPIs were extracted from its own architecture [29][30][33][34] (see subsection 2.4.2) and based on the number of occurrences of multi-cycle instructions performed for each application. The MIPS R3000 has the smallest and thus the best CPI compared to the other architectures whereas the MIPS24k has sometimes a greater CPI than DSR-HD. Indeed, the MIPS R3000—as a five-stage architecture—is made up of only the necessary functional stages (Instruction Fetch, Instruction Decode, ALU, MEM Access and WB): i.e., one stage per mandatory function. As a consequence, the number of *stall* cycles are reduced. On the other side, DSR-HD has greater CPIs than the MIPS32 24K, mainly because of memory instructions that cannot access directly the memories due to conflicts of ressources induced by the two clock domains described in chapter 5. On all the benchmarks, the greatest difference between DSR-HD and the MIPS R3000 CPIs is equal to 0.8, the smallest is equal to 0.03 and the average is equal to 0.39. The differences for the MIPS24k and the MIPS R3000 become 0.69, 0 and 0.22, respectively.

Tables C.1 and D.1 report the access time of the slowest memory (instruction or data memory) required for each EEMBC application (section 6.3 explains how

these values were obtained). Indeed, these access times depend mainly on the largest memory size, on their organization (32 or 64 bits) and on the various technology nodes. These timings allow us to accurately establish the maximum clock frequency of the whole system (see figure 7.1). Finally, this information, coupled with that of table B.1, leads to the performance of the studied processors for each EEMBC application. The access time grows when the transistor length decreases: this is due to the internal interconnections within the memory itself that scale more slowly than the transistors (details in subsection 6.3). Furthermore, the access time of a 64-bit memory is slower than that of a 32-bit memory—of the same capacity in terms of bytes—because its address decoder is smaller, and hence faster.

This section concerns the three EEMBC applications where DSR-HD obtained the best, the worst and a typical gain, under a conservative wire technology scaling, in terms of performance compared to the MIPS R3000 and the MIPS24k. The performance gain is the ratio of the difference between the DSR-HD and MIPS R3000 execution times divided by the MIPS R3000 execution time required to perform an EEMBC benchmark. The same approach has been applied to obtain the performance of the MIPS24k architecture compared to the R3000.

The core clock frequency was set to 330MHz for the MIPS R3000 processor instead of 350MHz as previously (explanation in table 2.1). This small difference may have an effect at the two first technology node levels (130 and 90nm): indeed, the core may be the critical path, like is the case in figure 7.9. In this condition, the EEMBC memories have no influence on the global clock frequency of the MIPS R3000.

The most common performance gain brought by DSR-HD over the MIPS R3000 is represented in figure 7.9. It corresponds to the conven00data_3 EEMBC program (a telecommunication application, see section 6.6). This typical gain[3] is about 32% in any technology node whereas the MIPS24k varies from 20 (130nm node) to -12.5% (22nm node). To make the analysis of this graph easier, figure 3.1 is summarized and updated in figure 7.10 according to the three well defined architectures mentioned so far.

An important information is that the maximum frequencies that DSR-HD and the MIPS24k can reach are assumed to be the same (explanation in section 5.5): so, $f_{2_{sat}} = f_{3_{sat}} = 625MHz$ whereas $f_{1_{sat}} = 330MHz$. We will note that the MIPS24k has a greatest performance gain than DSR-HD for a given frequency. Moreover, the highest

[3]More details on the EEMBC benchmarks in section 6.6.

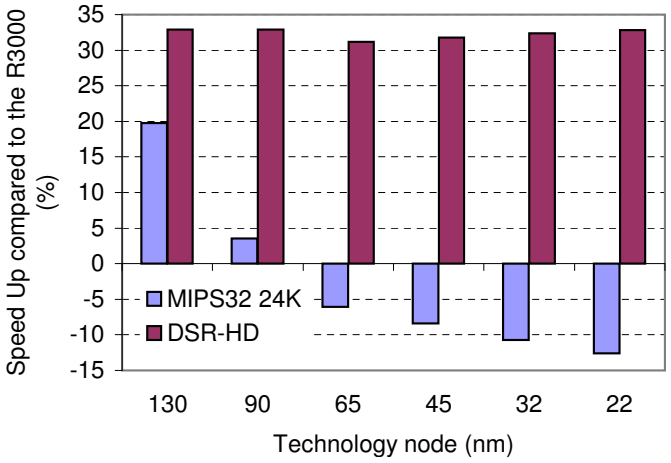

Figure 7.9: Typical DSR-HD and MIPS32 24K Performance Gains among all the EEMBC Applications: conven00data_3 Benchmark Results. The baseline is the execution time required by the MIPS R3000 processor with single port direct mapped primary memories to perform the conven00data_3 EEMBC benchmark. For each technology node, the DSR-HD and MIPS24k execution times are divided by this baseline.

performance is obtained by the five-stage processor when the frequency is inferior or equal to $f_{1_{sat}}$ due to its short pipeline length. Between $f_{1_{sat}}$ and $f_{1,2}$ ($f_{1,3}$), the MIPS24k (DSR-HD) is still lower performing than the R3000 with its $f_{1_{sat}}$ frequency because of a lower CPI. Once the frequency overcomes $f_{3,2}$, there is no way for DSR-HD to have a better performance than the MIPS24k because of its smaller CPI: however, this case never occurred.

In figure 7.9, we can observe that the gain of DSR-HD is constant during the two first technology nodes: indeed, both DSR-HD and the R3000 are not limited by the memory access time and may run at their maximum speed. On the other hand, the MIPS24k frequency is between $f_{1,2}$ and $f_{3,2}$ in the first node: that is the reason why the gain is positive but inferior to DSR-HD. So at this node, the MIPS24k is already limited by its instruction memory access time of just 4324 bytes (see table E.1). Consequently, its frequency will scale more slowly than that of the R3000 at 90nm because memories have a scale factor of about 10% higher than the raw logic: so, the MIPS24k speed increases but less than the R3000 and it remains located between the same range of frequencies than previously what still implies a positive gain.

As of the 65nm technology node, the memory becomes part of the critical path[4] of all the processors. The frequencies are below their maximum limit:

[4]This depends on the memory size required by an application and may occurs earlier or later than the 65nm node.

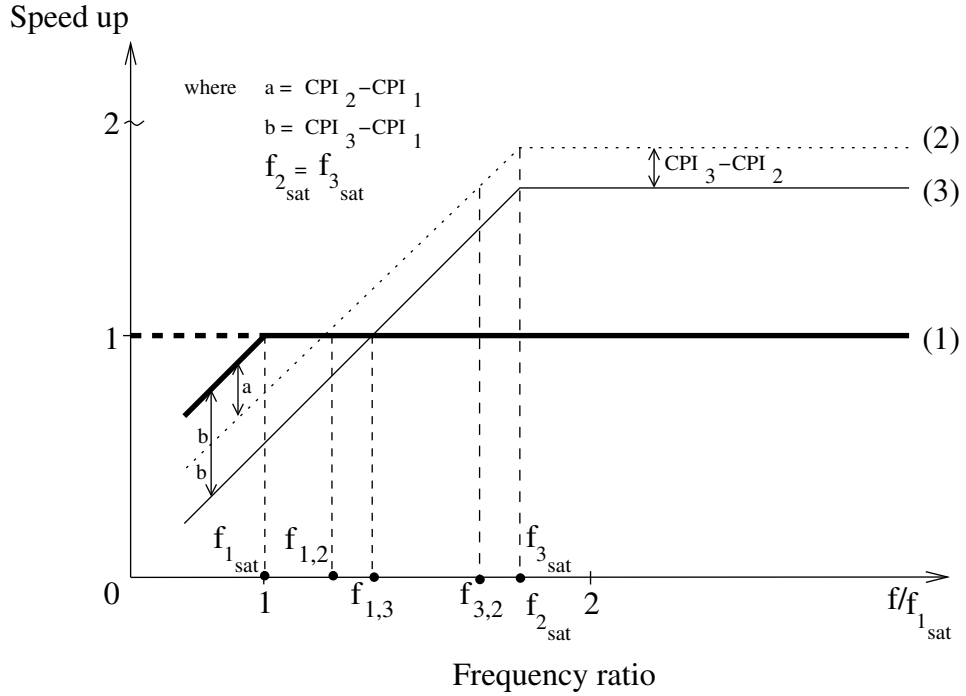

Figure 7.10: Expected Gain of DSR Compared to Scalar Architectures: $f_{i,j}$ is the threshold frequency that must be reached to pass from the architecture (i) to a more sophisticated architecture (j). (1) is the reference performance achieved by a five-stage processor like the MIPS R3000 architecture where the maximum frequency peaks at about 330 MHz ($f_{1_{sat}}$) in a 0.13μm High-performance process. (2) represents the gain in performance brought by a deeper pipeline like the last MIPS32 24K over a five-stage processor where the higher reachable frequency is $f_{2_{sat}}$. (3) corresponds to the gain in performance brought by the DSR architecture compared to (1) where the maximum frequency is $f_{3_{sat}}$.

i.e., $f_{1,3} < f_{DSR-HD} < f_{3_{sat}}$, $f_{1_{sat}} < f_{MIPS24k} < f_{1,2}$ and $f_{R3000} < f_{1_{sat}}$. So, the performance gain of the MIPS24k begins to be negative whereas the one of DSR-HD stays high at 31% but which is lower than in the previous node: this is due to its larger multiplexer that forwards the computed data back to the ALU stage entrance. Indeed, the critical path is the same except for this multiplexer, due the deeper pipeline. The frequency ratio has decreased to a value a bit inferior to $\frac{625MHz}{330MHz}$.

For the last three technology nodes, the DSR-HD gain increases slightly because of the interconnection delays that become significant as shown in figure 7.1, even if it is reduced by the forwarding multiplexer larger than the R3000's one. Indeed, the essence of DSR-HD is to be less sensitive to the delays of wires that connect the core to the primary memories, which is the case, as shown in figure 7.11: these gains

are the difference of gains normalized to the performance of the MIPS R3000 based on optimistic and pessimistic wire delay models. The graph highlights the effect of the deep-submicron wire delays on the performance of processors. The trend lines indicate that the gap of performance between the three processors increases as of the 65nm node.

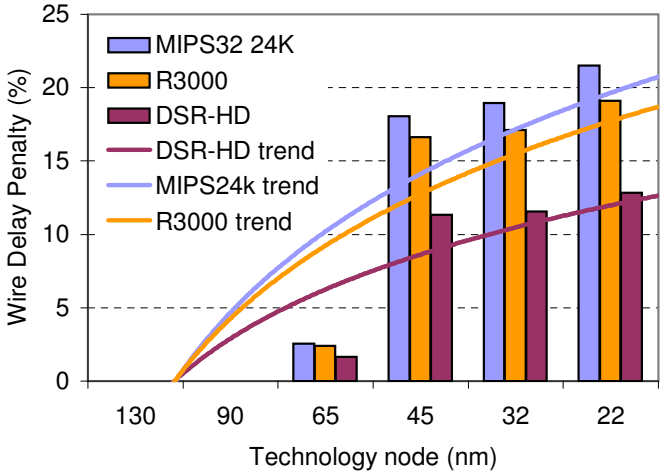

Figure 7.11: Wire Delay Penalty on the DSR-HD Performance Gain for the conven00data_3 Benchmark: the baseline is the execution time required by the MIPS R3000 processor with single port direct mapped primary memories to perform the conven00data_3 EEMBC benchmark. For each technology node, the DSR-HD and MIPS24k execution times are divided by the baseline.

On the MIPS24k side, the performance decreases for three reasons: the wire delays, the eight-stage's forwarding multiplexer and the slower memory access time scaling: a 64-bit memory access time is smaller than its 32-bit counterpart. However, the gap decreases with future technology nodes even if the scale factor is the same. Indeed, the scale factor applied to great number (e.g., a 32-bit access time) induces a greater absolute number than the same scaling factor applied to a smaller number (e.g., a 64-bit access time) as described in figure 7.12.

Figure 7.13 highlights the greatest performance gain of DSR-HD over the MIPS R3000: this is due to the fact that the data memory of this application is large—811KB (see table C.1)—and constitutes the speed bottleneck of the three architectures. All the frequencies are smaller than $f_{1_{sat}}$. The DSR-HD's frequency is greater than the frequency of the MIPS24k which is itself higher than the R3000 one. Furthermore, the CPIs are quite the same as reported in table B.1. That is the reason why all the

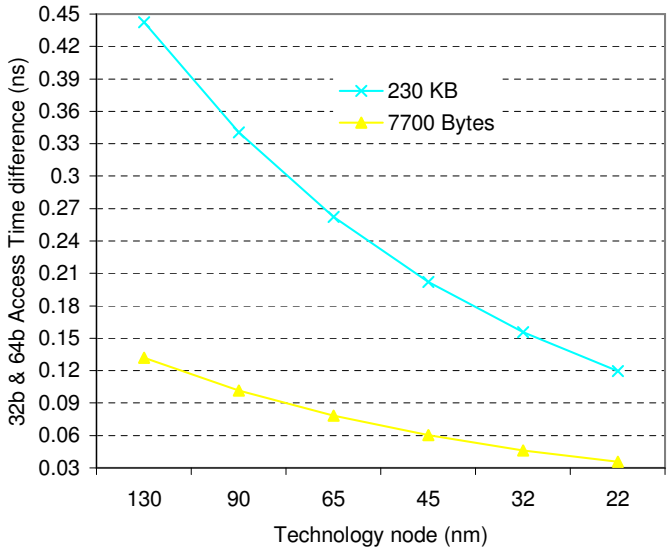

Figure 7.12: 32-bit vs. 64-bit HD Memory Access Time Trend: each line represents the difference of access time between a 32-bit and a 64-bit HD SRAM memory. The gap diminishes as gate widths get smaller.

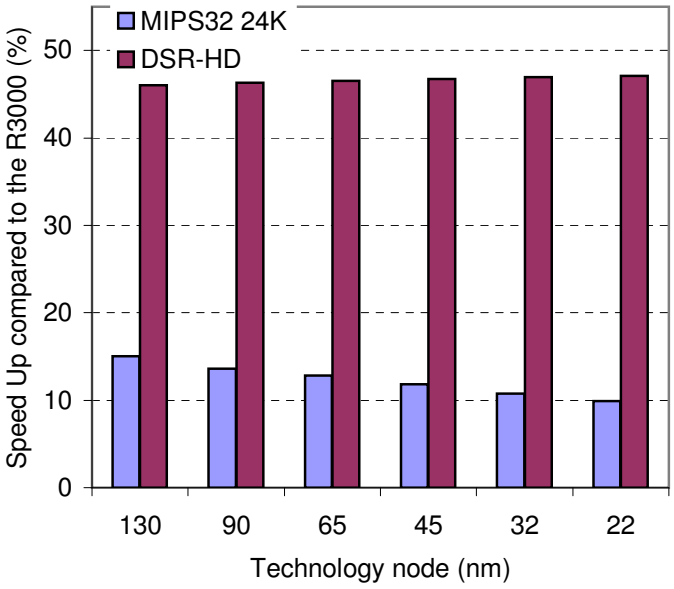

Figure 7.13: Best DSR-HD Performance Gains: rgbyiq01 Benchmark Results. The baseline is the execution time required by the MIPS R3000 processor with single port direct mapped primary memories to perform the rgbyiq01 EEMBC benchmark. For each technology node, the DSR-HD and MIPS24k execution time are divided by the baseline.

performance gains are positive. The explanations of their trends—that go from 15 to 10% for the MIPS24k and 46 to 47% for DSR-HD—are similar to those described for

figure 7.9.

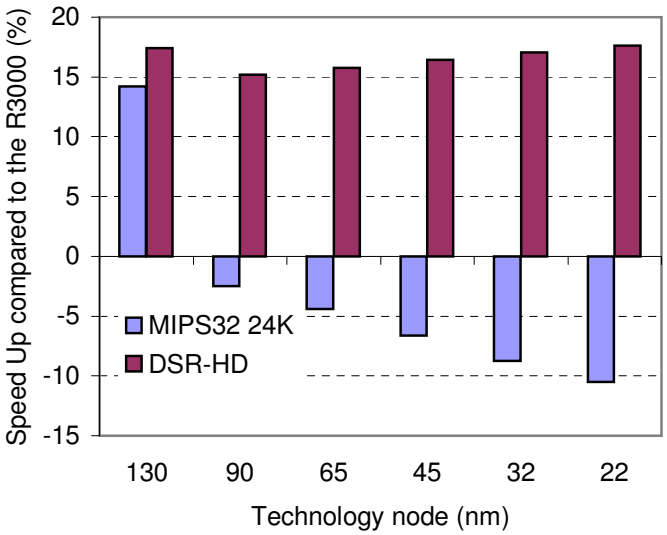

Figure 7.14: Worst DSR-HD Performance Gains: puwmod01 Benchmark Results. The baseline is the execution time required by the MIPS R3000 processor with single port direct mapped primary memories to perform the puwmod01 EEMBC benchmark. For each technology node, the DSR-HD and MIPS24k execution time are divided by the baseline.

Figure 7.14 shows the case where DSR-HD obtains the smallest gains compared to the R3000: The gains of the MIPS24k decrease from 14 to -10.5% whereas the gains are still positive for DSR-HD and are between 15 and 17.5%. The data memory size is equal to 9824 bytes are is part of the critical path as of the 90nm node.

The trends of gains are still the same under an aggressive wire technology scaling factor as shown by the the last three figures of appendix J. Except for the best case, the effect of the interconnections is delayed to 45nm node where the frequencies of the processors are limited by the primary memory access. The performance of the MIPS24k are better and those of DSR-HD are quite unchanged compared to the conservative model.

7.6 Scalar Embedded Processor Efficiency

So far, the benefit of DSR-HD compared to the MIPS R3000 and the the MIPS32 24k have been discussed and established in terms of frequency and performance. However, this is not a guarantee of quality: indeed, the use of a superscalar processor or a matrix

of processors would also lead to a greater performance than a scalar five-stage processor. But obviously, these solutions require more silicon area and consume more energy than a scalar processor which is not necessarily compatible with the embedded world. Moreover, these hardware solutions would be too sophisticated for many embedded applications. So, a key factor to determine the quality of an architecture and compare them is the ratio $\frac{Performance}{Area}$ (known as $\frac{MIPS}{Area}$), called *efficiency*.

The efficiency results were obtained under the same conditions as in the two previous sections. The surface of silicon involved in any processor has been introduced, and takes into account in the total area of the circuit: i.e., the raw core and the required primary memories. The R3000 and the MIPS24k core area used are 1.9mm^2 and 2.8mm^2, respectively [62]. Even if DSR-HD is not planned to be as sophisticated as the MIPS24k, the same surface of silicon will be considered.

According to Virage Logic memories [66] and CACTI3.0 report [44], the relation between SP SRAM memory capacity and their area is linear. So, the area of generated Virage Logic's SRAMs lead to the area of memories that require the EEMBC benchmarks, using the principle of linearity. Furthermore, the core and memory areas scale with a factor equal to 0.5 every technology node [60] (relative to the WxL gate size where W and L scale by a factor of 0.7). The 32-bit and 64-bit SP memory sizes used in this work are listed in appendixes E and F.

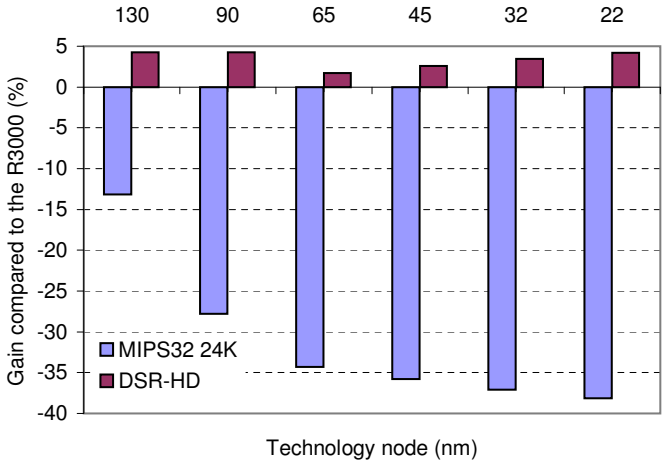

Figure 7.15: Typical DSR-HD and MIPS32 24K Efficiency Gains among all the EEMBC Applications: conven00data_3 Benchmark Results.

The most common gains of efficiency were observed for the same benchmark as for the average performance gains, i.e., conven00data_3, as shown in figure 7.15. The

monotony of efficiency gains follows precisely the one obtained for performance in figure 7.9 and then, obeys to the same explanation. However, they are reduced and vary from 1.7 to 4.3% for DSR-HD[5] and from -13% to -38% for the MIPS24k. This is due to the size of the memories required for this application that are small compared to the core sizes (see appendix F.1): the R3000 occupies 32% less silicon area than the two eight-stage processors.

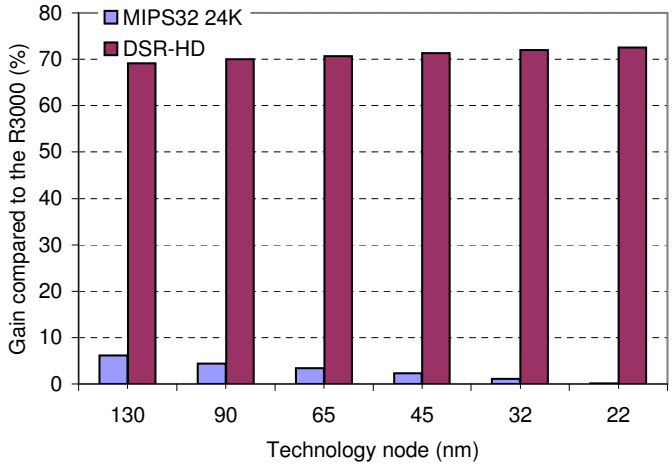

Figure 7.16: Best DSR-HD Efficiency Gains: rgbyiq01 Benchmark Results.

As for the typical efficiency gains, the benchmark corresponding to the best DSR-HD gain is the same as in the previous section and follows the same monotony. However, the gains stay important for DSR-HD and culminate at about 70% in any node as reported in figure 7.16: indeed, the rgbyiq01 program needs a large data memory (230 KBytes) which amplifies the efficiency result of DSR-HD to the detriment of the R3000. Likewise, the difference of gain between the MIPS24k and the R3000 (and DSR-HD as well) is reduced compared to the performance gain because the 64-bit memories occupy more space than their 32-bit counterparts (see appendix F.1): it goes from 6.1% to 0.1%.

The worst efficiency results are resumed in figure 7.17 where the DSR-HD efficiency is between -12.6 and -15% whereas the MIPS24k losses vary between -16.5% and -35.2%. Again, the benchmark and the gain monotony are similar to those of the worst performance case. All the gains are negative because the total memory area is small (about 12KB) compared to the core sizes and the performance gains were not important enough.

[5]DSR-HP occupies much more silicon area than DSR-HD due to its dual port primary memories which implies an insufficient efficiency gain

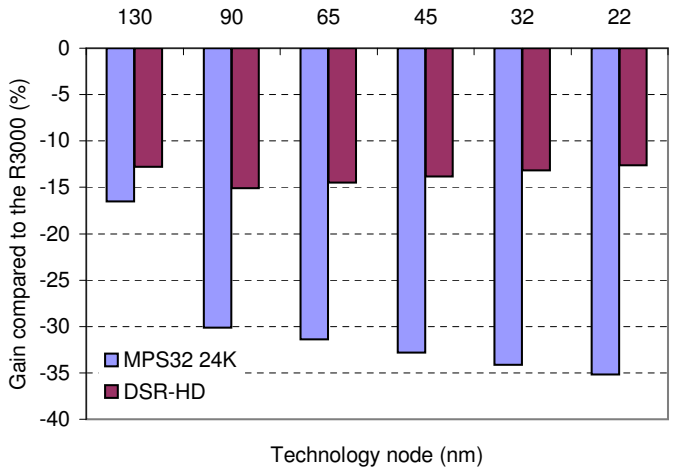

Figure 7.17: Worst DSR-HD Performance Gains: puwmod01 Benchmark Results.

When an optimistic wire technology scaling factor is taken into account the efficiency of the MIPS24k is slightly improved whereas the efficiency gains remain roughly unchanged as shown in appendix K. Indeed, DSR-HD is less sensitive to the wire delay since it is roughly divided by 2 compared to the R3000 and the 24k as written in equation 7.3.

7.7 Conclusion

In order to establish experimental results, the type of wires used to interconnect the primary memories and the core has been identified: the semi-global scaled-length wires—spanning 50K gates—have delays that can reach up to 30 or 53 FO4s in the 22nm technology node, depending if an aggressive or conservative prediction is considered.

First, it has been shown that the processor frequencies are increasingly limited by their primary memory access time coupled with the wire effect. Indeed, the larger the memory and smaller the core period can be. Likewise, the use of a High-density SRAM can reduce the CPU frequency because they are slower than their High-performance counterparts. Even if the MIPS24k and DSR-HD have the same intrinsic maximum speed (625MHz vs. 330MHz for the R3000), DSR-HD is the much less disturbed by slow memories and (conservative or aggressive) wire delays. In the worst case, it runs 15% faster than the MIPS R3000 in any technology node.

All three architectures are based on the same set of instructions, and are binary compatible allowing to perform the same program on every processor. Consequently, it was possible to compared both performance and efficiency of these architectures. The DSR-HD CPI is 0.4 greater than the R3000 one which is itself 0.2 smaller than the MIPS24k CPI, in average. However, taking clock frequency into account, it appears that DSR-HD is the most performing among the three processors: it is between 15 and 46% more performing than the R3000 in any technology node where, most of the time, the MIPS24k suffers of a negative gain compared to the R3000.

Finally, the efficiency parameter—defined as the $\frac{MIPS}{Area}$ ratio—was used to determine the best architecture. The benefit of the performance of DSR-HD is reduced in terms of efficiency when the total primary memory area is about the same than CPU cores: indeed, the R3000 core is 32% smaller than the eight-stage processor (the DSR-HD area has been aligned to the large MIPS24k size). The MIPS24 efficiency gain is negative most of time and goes from -13 to -38% in the typical case. The efficiency gain of DSR-HD varies between 1.7 and 4.3% in the typical case, between -12.6 and -15% in the worst case and constant at 70% in the best case, compared to the R3000.

Chapter 8

Conclusion

In this thesis, we explore the state-of-the-art of the most performing scalar embedded processor architectures like the ARM11, ARC700, Xtensa V, the PowerPC 405, and focus on the MIPS32 24k and the MIPS R3000. Three scalar architectures have been investigated to be independent of the foundry in deep-submicron technologies: DSR-HP to improve the performance, DSR-HD to improve the efficiency and to reduce the silicon area and DSR-LP to save energy.

The concern of this thesis work was to propose a scalar processor architecture more performing or/and that requires less silicon area than their counterparts. The efficiency criterion that takes into account both performance and area was chosen as the most relevant metric. Consequently, this thesis focused on the High Density version of the DSR architectures.

A simplified High-performance eight-stage architecture and a five-stage processor have been implemented in CL013LV-OD process from TSMC. Results like the critical paths, maximum frequencies and areas have been used to validate the proposed architectures.

8.1 Main Contributions

The main contribution of this work is the proposal of an embedded scalar processor architecture—DSR—that tackles the *Memory Wall*. The key idea is to allow two CPU clock cycles to access the primary memories instead of one, as is the case in all the previously cited processors. In other words, the primary memories run on a clock twice slower than the CPU clock. Thus, the effect of the memory access time

on the frequency of the processor is limited. Likewise, the delays of the wires that connect the core to its primary memories are reduced. As a consequence, and despite wasted cycles due to a deeper pipeline, DSR-HD is up to 72% more efficient than a standard five-stage processor and even more compared to the MIPS32 24k, which has the same intrinsic maximum frequency as DSR. The relative low efficiency of the 24k comes from the fact that the memory access time is still part of the critical path for future technology nodes and its deeper pipeline implies extra latencies compared to a five-stage scalar processor. Furthermore, the new MIPS24k's functions, implemented to improve performance, were not used to remain binary compatible with the R3000 and DSR. Finally, the 24k occupies more area than the R3000, mainly due to its extra functions. So, the slower the memory path, the more benefit DSR has compared to other scalar architectures.
Furthermore, DSR is efficient in various domains of applications.

The second contribution is the study of the effect of both primary memory and interconnection delays on scalar embedded processor frequencies as a function of technology nodes.

A cache model—CACTI—, the ITRS roadmap and a Virage Logic's SRAM compiler were used to calculate and validate industrial memory access times and areas. Thus, various HD SRAM sizes (especially those required by the EEMBC benchmarks) and their corresponding access time and area, for several technology nodes, have been listed for both 32 and 64-bit word width.

Finally, a virtual operating system has been implemented on the five-stage simulator VMIPS that allows to run any program, even those that use standard I/O functions. A profiler has also been attached in order to extract many statistic on the behavior of C programs. This last point allowed to establish and report MIPS R3000 CPIs associated to the EEMBC benchmarks, as well as the number of instructions executed.

8.2 Perspectives

The reduction of power consumption suggested by DSR-LP, due to a lower memory power supply voltage, should be investigated in detail and applied to the EEMBC benchmarks for several technology nodes. The partitioning of the static and dynamic power consumption between the processor core and its primary memories according to the number of read and write operations should be identified. Then, the formula giving the new power supply of the memories as a function of the reduced memory frequency has to be established.

For a performance point of view, on-chip memory re-sizing should be studied by limiting the number of cache misses without deteriorating the clock period. Thus, the increase in the size of the caches that does not reduce the frequency of the processor has to be determine. In this case, only applications that require caches and where DSR runs at its maximum intrinsic speed should be taken into account.

Appendix A

Aggressive Embedded Processor Clock Period Scaling

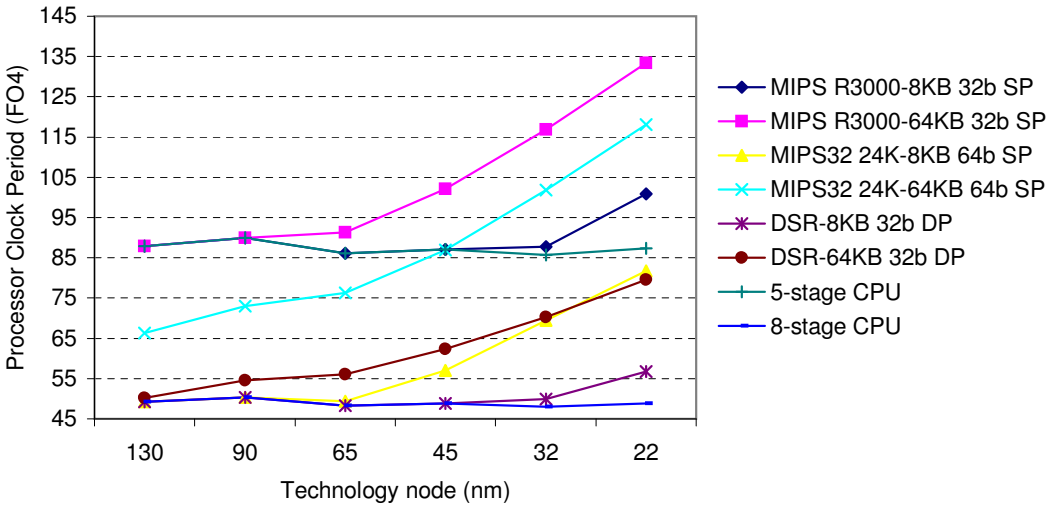

Figure A.1: HS Embedded Processor Clock Period Scaling: three processor clock period scalings are reported in this graph including direct mapped primary memory access time, forwarding multiplexer, aggressive wires and clock overhead. The bare CPU is no longer the critical path as of the 32nm node for the five-stage processor with a (small) 8KB data memory, whereas the same path already limits the frequency at the 65nm node with a 64KB memory. The MIPS24K clock period is smallest but has roughly the same behavior whereas DSR-HP is less sensitive to technology scaling and limits its effect for later technology nodes.

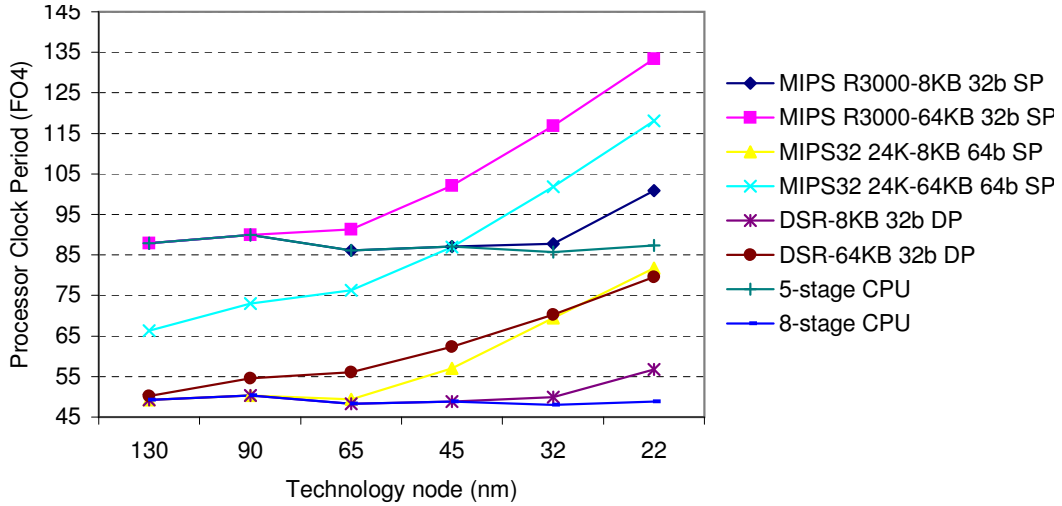

Figure A.2: HD Embedded Processor Clock Period Scaling: three processor clock period scalings are reported in this graph including direct mapped primary memory access time, forwarding multiplexer, aggressive wires and clock overhead. The bare CPU is no longer the critical path as of the 45nm node for the five-stage processor with a (small) 8KB data memory, whereas the same path already limits the frequency at the 130nm node with a 64KB memory. The DSR-HD is less sensitive to technology scaling and limits its effect for later technology nodes. In any case, DSR-HD proposes the fastest clock frequency of complete micro-processors.

Appendix B

EEMBC Benchmark CPIs

		CPI			
EEMBC benchmark	**nbr.inst**	**MIPS R3000**	**MIPS24K**	**DSR-HP**	**DSR-HD**
routelookup	24434068	1.24	1.57	1.69	1.8
pktflowb512k	6866547	1.09	1.41	1.5	1.62
pktflowb1m	13175661	1.09	1.42	1.49	1.62
pktflowb2m	25793422	1.09	1.42	1.49	1.62
pktflowb4m	51539785	1.09	1.42	1.49	1.62
ospf	6007682	1.38	2.07	1.92	2.05
viterb00data_1	736777382	1.02	1.16	1.28	1.36
viterb00data_2	737005548	1.02	1.16	1.28	1.36
viterb00data_3	734527570	1.02	1.18	1.28	1.36
fft00data_1	49726580	1.05	1.2	1.22	1.26
fft00data_2	49726290	1.05	1.2	1.22	1.26
fft00data_3	49724064	1.05	1.2	1.22	1.26
fbital00data_2	1845092275	1.09	1.36	1.42	1.5
fbital00data_3	127575142	1.16	1.44	1.47	1.55
fbital00data_6	1218347958	1.1	1.37	1.4	1.49
conven00data_1	499748024	1.09	1.34	1.38	1.42
conven00data_2	435713061	1.09	1.32	1.38	1.42
conven00data_3	351198296	1.08	1.36	1.35	1.38
autcor00data_1	5686842	1.09	1.13	1.39	1.44
autcor00data_2	814451733	1.1	1.1	1.42	1.47
autcor00data_3	777574731	1.1	1.1	1.42	1.47
viterb00data_4	742077842	1.01	1.13	1.28	1.36
bezier01fixed	694462353	1	1.07	1.1	1.17
bezier01float	694462353	1	1.07	1.1	1.17
dither01	2386029832	1.08	1.42	1.38	1.44
rotate01	560117123	1.04	1.43	1.5	1.55
text01	1082934918	1.1	1.5	1.46	1.58
rgbyiq01	668870479	1	1.04	1.02	1.03
rgbhpg01	367000642	1	1.01	1.18	1.21
rgbcmy01	2680307236	1	1.21	1.1	1.13
djpeg	17519804965	1.02	1.31	1.25	1.31
cjpeg	21198012938	1.05	1.32	1.34	1.42
ttsprk01	49432900	1.21	1.52	1.57	1.65
tblook01	17305460	1.22	1.76	1.62	1.74
rspeed01	28295001	1.1	1.33	1.54	1.66
puwmod01	40667434	1.1	1.37	1.59	1.73
pntrch01	75645765	1.51	1.87	2.22	2.31
canrdr01	46076248	1.06	1.35	1.43	1.53
cacheb01	30721150	1.07	1.31	1.42	1.53
bitmnp01	237244195	1.22	1.58	1.72	1.78
aifirf01	54834193	1.08	1.14	1.39	1.6
a2time01	30153900	1.38	1.56	1.83	1.93

Table B.1: CPIs as a function of embedded architectures for EEMBC benchmarks: for any EEMBC application, MIPS R3000 has the smallest and then the best CPI compared to the other architectures whereas the MIPS24k has sometimes a greater CPI than DSR-HD. This table reports also the number of instructions performed for each applications: these numbers are exactly those executed by the MIPS R3000 and have been kept unchanged for the other processors as explained in chapter 7.1.

Appendix C

32-bit HD Memory Access Time

Program name	Max(iram,dram) size (Bytes)	HD SP 32b Memory Access time (fo4)					
		130nm	90nm	65nm	45nm	32nm	22nm
ospf	7700	49.9	56.2	59.2	65.9	71.3	79.9
pktflowb512k	4604	46.1	51.8	54.6	60.8	65.8	73.7
pktflowb1m	4604	46.1	51.8	54.6	60.8	65.8	73.7
pktflowb2m	4604	46.1	51.8	54.6	60.8	65.8	73.7
pktflowb4m	4604	46.1	51.8	54.6	60.8	65.8	73.7
routelookup	9160	51.2	57.6	60.7	67.6	73.2	82
autcor00data_1	3768	42	47.2	49.8	55.4	60	67.2
autcor00data_2	3768	42	47.2	49.8	55.4	60	67.2
autcor00data_3	3768	42	47.2	49.8	55.4	60	67.2
conven00data_1	4324	46	51.7	54.6	60.7	65.7	73.6
conven00data_2	4324	46	51.7	54.6	60.7	65.7	73.6
conven00data_3	4324	46	51.7	54.6	60.7	65.7	73.6
fbital00data_2	4120	45.9	51.6	54.4	60.5	65.5	73.4
fbital00data_3	4120	45.9	51.6	54.4	60.5	65.5	73.4
fbital00data_6	4120	45.9	51.6	54.4	60.5	65.5	73.4
fft00data_1	6676	48.8	54.9	57.9	64.4	69.7	78.1
fft00data_2	6676	48.8	54.9	57.9	64.4	69.7	78.1
fft00data_3	5072	46.7	52.5	55.3	61.6	66.7	74.7
viterb00data_1	4652	46.5	52.3	55.1	61.3	66.4	74.3
viterb00data_2	4652	46.5	52.3	55.1	61.3	66.4	74.3
viterb00data_3	4652	46.5	52.3	55.1	61.3	66.4	74.3
viterb00data_4	4652	46.5	52.3	55.1	61.3	66.4	74.3
bezier01fixed	19176	61.7	69.4	73.1	81.4	88.1	98.7
bezier01float	19176	61.7	69.4	73.1	81.4	88.1	98.7
dither01	65632	78.9	88.7	93.5	104.1	112.7	126.2
rotate01	25900	59.9	67.4	71.1	79	85.6	95.9
text01	57812	76.6	86.1	90.8	101	109.4	122.5
cjpeg	762760	220.2	247.7	261.1	290.4	314.5	352.2
djpeg	811968	224.5	252.5	266.2	296.1	320.6	359.1
rgbcmy01	230824	126.4	142.2	149.9	166.8	180.6	202.3
rgbhpg01	77224	61.7	94.1	99.2	110.3	119.4	133.8
rgbyiq01	230824	126.4	142.2	149.9	166.8	180.6	202.3
ttsprk01	49844	72.3	81.3	85.7	95.4	103.3	115.7
tblook01	12332	54.8	61.7	65	72.3	78.3	87.7
rspeed01	2084	40.1	45.2	47.6	53	57.4	64.2
puwmod01	9824	52.2	58.7	61.9	68.8	74.5	83.5
pntrch01	5516	47.2	53.1	56	62.3	67.5	75.6
idctrn01	12140	54.4	61.2	64.5	71.8	77.7	87
canrdr01	6088	47.8	53.8	56.7	63.1	68.3	76.6
cacheb01	4212	45.9	51.6	54.4	60.5	65.5	73.4
bitmnp01	10800	53.2	59.9	63.1	70.2	76	85.2
basefp01	16660	59.1	66.5	70.1	78	84.4	94.6
aiifft01	44180	69.6	78.2	82.5	91.8	99.4	111.3
aifirf01	4400	46	51.7	54.6	60.7	65.7	73.6
aifftr01	44180	69.6	78.2	82.5	91.8	99.4	111.3
a2time01	4432	46	51.7	54.6	60.7	65.7	73.6

Table C.1: 32-bit HD Memory Access Time for EEMBC Benchmarks: each memory size represents the largest SP memory among the instruction and data memories. In any case, the access time grows when the transistor length decreases: this is due to the internal interconnections within the memory itself that scale slower than the transistors (see subsection 6.3). These access times were obtained as described in subsection 6.3.

Appendix D

64-bit HD Memory Access Time

Program name	Max(iram,dram) size (Bytes)	HD SP 64b Memory Access time (fo4)					
		130nm	90nm	65nm	45nm	32nm	22nm
ospf	7700	45.9	51.6	54.4	60.5	65.5	73.4
pktflowb512k	4604	45.7	51.4	54.2	60.3	65.3	73.2
pktflowb1m	4604	45.7	51.4	54.2	60.3	65.3	73.2
pktflowb2m	4604	45.7	51.4	54.2	60.3	65.3	73.2
pktflowb4m	4604	45.7	51.4	54.2	60.3	65.3	73.2
routelookup	9160	51	57.3	60.4	67.2	72.8	81.6
autcor00data_1	3768	41.7	46.9	49.4	55	59.5	66.7
autcor00data_2	3768	41.7	46.9	49.4	55	59.5	66.7
autcor00data_3	3768	41.7	46.9	49.4	55	59.5	66.7
conven00data_1	4324	45.7	51.4	54.2	60.3	65.2	73.1
conven00data_2	4324	45.7	51.4	54.2	60.3	65.2	73.1
conven00data_3	4324	45.7	51.4	54.2	60.3	65.2	73.1
fbital00data_2	4120	45.7	51.4	54.2	60.3	65.2	73.1
fbital00data_3	4120	45.7	51.4	54.2	60.3	65.2	73.1
fbital00data_6	4120	45.7	51.4	54.2	60.3	65.2	73.1
fft00data_1	6676	45.3	50.9	53.7	59.7	64.7	72.5
fft00data_2	6676	45.3	50.9	53.7	59.7	64.7	72.5
fft00data_3	5072	46.4	52.1	55	61.1	66.2	74.2
viterb00data_1	4652	46.1	51.9	54.7	60.9	65.9	73.8
viterb00data_2	4652	46.1	51.9	54.7	60.9	65.9	73.8
viterb00data_3	4652	46.1	51.9	54.7	60.9	65.9	73.8
viterb00data_4	4652	46.1	51.9	54.7	60.9	65.9	73.8
bezier01fixed	19176	55.9	62.9	66.3	73.7	79.9	89.5
bezier01float	19176	55.9	62.9	66.3	73.7	79.9	89.5
dither01	65632	78.4	88.2	93	103.5	112	125.5
rotate01	25900	59.7	67.2	70.8	78.8	85.3	95.5
text01	57812	76.4	85.9	90.6	100.7	109.1	122.2
cjpeg	762760	218.4	245.6	259	288	311.9	349.3
djpeg	811968	222.6	250.4	264	293.7	318	356.1
rgbcmy01	230824	112.8	126.9	133.8	148.8	161.2	180.5
rgbhpg01	77224	55.9	93.5	98.6	109.7	118.8	133.1
rgbyiq01	230824	112.8	126.9	133.8	148.8	161.2	180.5
ttsprk01	49844	72.1	81.1	85.5	95.1	103	115.4
tblook01	12332	52.3	58.8	62	69	74.7	83.6
rspeed01	2084	39.8	44.7	47.2	52.4	56.8	63.6
puwmod01	9824	50.5	56.8	59.9	66.7	72.2	80.9
pntrch01	5516	46.9	52.8	55.7	61.9	67	75.1
idctrn01	12140	51.8	58.3	61.5	68.4	74.1	82.9
canrdr01	6088	44.8	50.4	53.1	59.1	64	71.6
cacheb01	4212	45.5	51.2	54	60.1	65.1	72.9
bitmnp01	10800	51.2	57.6	60.7	67.5	73.1	81.9
basefp01	16660	54.7	61.5	64.9	72.1	78.1	87.5
aiifft01	44180	69.4	78	82.3	91.5	99.1	111
aifirf01	4400	45.7	51.4	54.2	60.3	65.2	73.1
aifftr01	44180	69.4	78	82.3	91.5	99.1	111
a2time01	4432	45.7	51.4	54.2	60.3	65.2	73.1

Table D.1: 64-bit HD Memory Access Time for EEMBC Benchmarks: each memory size represents the largest SP memory among the instruction and data memories. In any case, the access time grows when the transistor length decreases: this is due to the internal interconnections within the memory itself that scale slower than the transistors (see subsection 6.3). These access times—obtained as described in subsection 6.3— are smaller than their 32-bit counterparts due to a smaller size (surface) for a same capacity (bytes) because of address port and hence the address decoder that are smaller.

Appendix E

HD Level 1 Instruction Memory Feature

EEMBC benchmark	Processor configuration	Inst. SRAM size (Bytes)	Instruction SRAM area (sq.um)					
			130nm	90nm	65nm	45nm	32nm	22nm
ospf	R3000	7700	262355	131177	65588	32794	16397	8198
	MIPS24K		279225	139612	69806	34903	17451	8725
	DSR-HD		279225	139612	69806	34903	17451	8725
pktflowb512k	R3000	2608	95794	47897	23948	11974	5987	2993
	MIPS24K		110265	55132	27566	13783	6891	3445
	DSR-HD		110265	55132	27566	13783	6891	3445
pktflowb1m	R3000	2620	96186	48093	24046	12023	6011	3005
	MIPS24K		110663	55331	27665	13832	6916	3458
	DSR-HD		110663	55331	27665	13832	6916	3458
pktflowb2m	R3000	2620	96186	48093	24046	12023	6011	3005
	MIPS24K		110663	55331	27665	13832	6916	3458
	DSR-HD		110663	55331	27665	13832	6916	3458
pktflowb4m	R3000	2620	96186	48093	24046	12023	6011	3005
	MIPS24K		110663	55331	27665	13832	6916	3458
	DSR-HD		110663	55331	27665	13832	6916	3458
routelookup	R3000	2720	99457	49728	24864	12432	6216	3108
	MIPS24K		113981	56990	28495	14247	7123	3561
	DSR-HD		113981	56990	28495	14247	7123	3561
autcor00data_1	R3000	3768	133738	66869	33434	16717	8358	4179
	MIPS24K		148755	74377	37188	18594	9297	4648
	DSR-HD		148755	74377	37188	18594	9297	4648
autcor00data_2	R3000	3768	133738	66869	33434	16717	8358	4179
	MIPS24K		148755	74377	37188	18594	9297	4648
	DSR-HD		148755	74377	37188	18594	9297	4648
autcor00data_3	R3000	3768	133738	66869	33434	16717	8358	4179
	MIPS24K		148755	74377	37188	18594	9297	4648
	DSR-HD		148755	74377	37188	18594	9297	4648
conven00data_1	R3000	4324	151925	75962	37981	18990	9495	4747
	MIPS24K		167204	83602	41801	20900	10450	5225
	DSR-HD		167204	83602	41801	20900	10450	5225
conven00data_2	R3000	4324	151925	75962	37981	18990	9495	4747
	MIPS24K		167204	83602	41801	20900	10450	5225
	DSR-HD		167204	83602	41801	20900	10450	5225
conven00data_3	R3000	4324	151925	75962	37981	18990	9495	4747
	MIPS24K		167204	83602	41801	20900	10450	5225
	DSR-HD		167204	83602	41801	20900	10450	5225
fbital00data_2	R3000	4120	145252	72626	36313	18156	9078	4539
	MIPS24K		160435	80217	40108	20054	10027	5013
	DSR-HD		160435	80217	40108	20054	10027	5013
fbital00data_3	R3000	4120	145252	72626	36313	18156	9078	4539
	MIPS24K		160435	80217	40108	20054	10027	5013
	DSR-HD		160435	80217	40108	20054	10027	5013
fbital00data_6	R3000	4120	145252	72626	36313	18156	9078	4539
	MIPS24K		160435	80217	40108	20054	10027	5013
	DSR-HD		160435	80217	40108	20054	10027	5013
fft00data_1	R3000	5072	176392	88196	44098	22049	11024	5512
	MIPS24K		192024	96012	48006	24003	12001	6000
	DSR-HD		192024	96012	48006	24003	12001	6000
fft00data_2	R3000	5072	176392	88196	44098	22049	11024	5512
	MIPS24K		192024	96012	48006	24003	12001	6000
	DSR-HD		192024	96012	48006	24003	12001	6000
fft00data_3	R3000	5072	176392	88196	44098	22049	11024	5512
	MIPS24K		192024	96012	48006	24003	12001	6000
	DSR-HD		192024	96012	48006	24003	12001	6000
viterb00data_1	R3000	4652	162654	81327	40663	20331	10165	5082
	MIPS24K		178088	89044	44522	22261	11130	5565
	DSR-HD		178088	89044	44522	22261	11130	5565
viterb00data_2	R3000	4652	162654	81327	40663	20331	10165	5082
	MIPS24K		178088	89044	44522	22261	11130	5565
	DSR-HD		178088	89044	44522	22261	11130	5565
viterb00data_3	R3000	4652	162654	81327	40663	20331	10165	5082
	MIPS24K		178088	89044	44522	22261	11130	5565
	DSR-HD		178088	89044	44522	22261	11130	5565
viterb00data_4	R3000	4652	162654	81327	40663	20331	10165	5082
	MIPS24K		178088	89044	44522	22261	11130	5565
	DSR-HD		178088	89044	44522	22261	11130	5565

EEMBC benchmark	Processor configuration	Inst. SRAM size (Bytes)	Instruction SRAM area (sq.um)					
			130nm	90nm	65nm	45nm	32nm	22nm
bezier01fixed	R3000	1212	50130	25065	12532	6266	3133	1566
	MIPS24K		63944	31972	15986	7993	3996	1998
	DSR-HD		63944	31972	15986	7993	3996	1998
bezier01float	R3000	1212	50130	25065	12532	6266	3133	1566
	MIPS24K		63944	31972	15986	7993	3996	1998
	DSR-HD		63944	31972	15986	7993	3996	1998
dither01	R3000	1008	43457	21728	10864	5432	2716	1358
	MIPS24K		57175	28587	14293	7146	3573	1786
	DSR-HD		57175	28587	14293	7146	3573	1786
rotate01	R3000	2948	106915	53457	26728	13364	6682	3341
	MIPS24K		121547	60773	30386	15193	7596	3798
	DSR-HD		121547	60773	30386	15193	7596	3798
text01	R3000	3624	129027	64513	32256	16128	8064	4032
	MIPS24K		143977	71988	35994	17997	8998	4499
	DSR-HD		143977	71988	35994	17997	8998	4499
cjpeg	R3000	51216	1685784	842892	421446	210723	105361	52680
	MIPS24K		1723146	861573	430786	215393	107696	53848
	DSR-HD		1723146	861573	430786	215393	107696	53848
djpeg	R3000	51020	1679373	839686	419843	209921	104960	52480
	MIPS24K		1716642	858321	429160	214580	107290	53645
	DSR-HD		1716642	858321	429160	214580	107290	53645
rgbcmy01	R3000	1232	50784	25392	12696	6348	3174	1587
	MIPS24K		64607	32303	16151	8075	4037	2018
	DSR-HD		64607	32303	16151	8075	4037	2018
rgbhpg01	R3000	1412	56672	28336	14168	7084	3542	1771
	MIPS24K		70580	35290	17645	8822	4411	2205
	DSR-HD		70580	35290	17645	8822	4411	2205
rgbyiq01	R3000	1392	56018	28009	14004	7002	3501	1750
	MIPS24K		69916	34958	17479	8739	4369	2184
	DSR-HD		69916	34958	17479	8739	4369	2184
ttsprk01	R3000	8312	282374	141187	70593	35296	17648	8824
	MIPS24K		299532	149766	74883	37441	18720	9360
	DSR-HD		299532	149766	74883	37441	18720	9360
tblook01	R3000	3064	110710	55355	27677	13838	6919	3459
	MIPS24K		125396	62698	31349	15674	7837	3918
	DSR-HD		125396	62698	31349	15674	7837	3918
rspeed01	R3000	1868	71588	35794	17897	8948	4474	2237
	MIPS24K		85711	42855	21427	10713	5356	2678
	DSR-HD		85711	42855	21427	10713	5356	2678
puwmod01	R3000	3456	123532	61766	30883	15441	7720	3860
	MIPS24K		138403	69201	34600	17300	8650	4325
	DSR-HD		138403	69201	34600	17300	8650	4325
pntrch01	R3000	2740	100111	50055	25027	12513	6256	3128
	MIPS24K		114645	57322	28661	14330	7165	3582
	DSR-HD		114645	57322	28661	14330	7165	3582
canrdr01	R3000	2584	95009	47504	23752	11876	5938	2969
	MIPS24K		109469	54734	27367	13683	6841	3420
	DSR-HD		109469	54734	27367	13683	6841	3420
cacheb01	R3000	4212	148261	74130	37065	18532	9266	4633
	MIPS24K		163488	81744	40872	20436	10218	5109
	DSR-HD		163488	81744	40872	20436	10218	5109
bitmnp01	R3000	10800	363758	181879	90939	45469	22734	11367
	MIPS24K		382087	191043	95521	47760	23880	11940
	DSR-HD		382087	191043	95521	47760	23880	11940
aifirf01	R3000	3400	121700	60850	30425	15212	7606	3803
	MIPS24K		136545	68272	34136	17068	8534	4267
	DSR-HD		136545	68272	34136	17068	8534	4267
a2time01	R3000	4432	155457	77728	38864	19432	9716	4858
	MIPS24K		170788	85394	42697	21348	10674	5337
	DSR-HD		170788	85394	42697	21348	10674	5337

Table E.1: Instruction Memory Areas for EEMBC Applications: this table reports the silicon area of the primary memories required for the MIPS R3000, the MIPS24k and DSR-HD architectures. All memories were generated in a 130nm technology (more details in subsection 6.3): the surface to the next technology node was extrapolated by applying a multiplying factor of 0.5 as mentioned in [60].

Appendix F

HD Level 1 Data Memory Feature

EEMBC benchmark	Processor configuration	Data SRAM size (Bytes)	Data SRAM area (sq.um)					
			130nm	90nm	65nm	45nm	32nm	22nm
ospf	R3000	772	35737	17868	8934	4467	2233	1116
	MIPS24K		49344	24672	12336	6168	3084	1542
	DSR-HD		35737	17868	8934	4467	2233	1116
pktflowb512k	R3000	4604	161084	80542	40271	20135	10067	5033
	MIPS24K		176495	88247	44123	22061	11030	5515
	DSR-HD		161084	80542	40271	20135	10067	5033
pktflowb1m	R3000	4604	161084	80542	40271	20135	10067	5033
	MIPS24K		176495	88247	44123	22061	11030	5515
	DSR-HD		161084	80542	40271	20135	10067	5033
pktflowb2m	R3000	4604	161084	80542	40271	20135	10067	5033
	MIPS24K		176495	88247	44123	22061	11030	5515
	DSR-HD		161084	80542	40271	20135	10067	5033
pktflowb4m	R3000	4604	161084	80542	40271	20135	10067	5033
	MIPS24K		176495	88247	44123	22061	11030	5515
	DSR-HD		161084	80542	40271	20135	10067	5033
routelookup	R3000	9160	310113	155056	77528	38764	19382	9691
	MIPS24K		327669	163834	81917	40958	20479	10239
	DSR-HD		310113	155056	77528	38764	19382	9691
autcor00data_1	R3000	612	30503	15251	7625	3812	1906	953
	MIPS24K		44035	22017	11008	5504	2752	1376
	DSR-HD		30503	15251	7625	3812	1906	953
autcor00data_2	R3000	2692	98541	49270	24635	12317	6158	3079
	MIPS24K		113052	56526	28263	14131	7065	3532
	DSR-HD		98541	49270	24635	12317	6158	3079
autcor00data_3	R3000	1772	68448	34224	17112	8556	4278	2139
	MIPS24K		82525	41262	20631	10315	5157	2578
	DSR-HD		68448	34224	17112	8556	4278	2139
conven00data_1	R3000	2132	80223	40111	20055	10027	5013	2506
	MIPS24K		94471	47235	23617	11808	5904	2952
	DSR-HD		80223	40111	20055	10027	5013	2506
conven00data_2	R3000	2132	80223	40111	20055	10027	5013	2506
	MIPS24K		94471	47235	23617	11808	5904	2952
	DSR-HD		80223	40111	20055	10027	5013	2506
conven00data_3	R3000	2132	80223	40111	20055	10027	5013	2506
	MIPS24K		94471	47235	23617	11808	5904	2952
	DSR-HD		80223	40111	20055	10027	5013	2506
fbital00data_2	R3000	2716	99326	49663	24831	12415	6207	3103
	MIPS24K		113849	56924	28462	14231	7115	3557
	DSR-HD		99326	49663	24831	12415	6207	3103
fbital00data_3	R3000	1780	68709	34354	17177	8588	4294	2147
	MIPS24K		82791	41395	20697	10348	5174	2587
	DSR-HD		68709	34354	17177	8588	4294	2147
fbital00data_6	R3000	2100	79177	39588	19794	9897	4948	2474
	MIPS24K		93409	46704	23352	11676	5838	2919
	DSR-HD		79177	39588	19794	9897	4948	2474
fft00data_1	R3000	6676	228860	114430	57215	28607	14303	7151
	MIPS24K		245247	122623	61311	30655	15327	7663
	DSR-HD		228860	114430	57215	28607	14303	7151
fft00data_2	R3000	6676	228860	114430	57215	28607	14303	7151
	MIPS24K		245247	122623	61311	30655	15327	7663
	DSR-HD		228860	114430	57215	28607	14303	7151
fft00data_3	R3000	4628	161869	80934	40467	20233	10116	5058
	MIPS24K		177291	88645	44322	22161	11080	5540
	DSR-HD		161869	80934	40467	20233	10116	5058
viterb00data_1	R3000	4364	153233	76616	38308	19154	9577	4788
	MIPS24K		168531	84265	42132	21066	10533	5266
	DSR-HD		153233	76616	38308	19154	9577	4788
viterb00data_2	R3000	4372	153495	76747	38373	19186	9593	4796
	MIPS24K		168797	84398	42199	21099	10549	5274
	DSR-HD		153495	76747	38373	19186	9593	4796
viterb00data_3	R3000	4364	153233	76616	38308	19154	9577	4788
	MIPS24K		168531	84265	42132	21066	10533	5266
	DSR-HD		153233	76616	38308	19154	9577	4788
viterb00data_4	R3000	4372	153495	76747	38373	19186	9593	4796
	MIPS24K		168797	84398	42199	21099	10549	5274
	DSR-HD		153495	76747	38373	19186	9593	4796

EEMBC benchmark	Processor configuration	Data SRAM size (Bytes)	Data SRAM area (sq.um)					
			130nm	90nm	65nm	45nm	32nm	22nm
bezier01fixed	R3000	19176	637741	318870	159435	79717	39858	19929
	MIPS24K		660014	330007	165003	82501	41250	20625
	DSR-HD		637741	318870	159435	79717	39858	19929
bezier01float	R3000	19176	637741	318870	159435	79717	39858	19929
	MIPS24K		660014	330007	165003	82501	41250	20625
	DSR-HD		637741	318870	159435	79717	39858	19929
dither01	R3000	65632	2157338	1078669	539334	269667	134833	67416
	MIPS24K		2201489	1100744	550372	275186	137593	68796
	DSR-HD		2157338	1078669	539334	269667	134833	67416
rotate01	R3000	25900	857686	428843	214421	107210	53605	26802
	MIPS24K		883126	441563	220781	110390	55195	27597
	DSR-HD		857686	428843	214421	107210	53605	26802
text01	R3000	57812	1901542	950771	475385	237692	118846	59423
	MIPS24K		1942010	971005	485502	242751	121375	60687
	DSR-HD		1901542	950771	475385	237692	118846	59423
cjpeg	R3000	762760	24960719	12480359	6240179	3120089	1560044	780022
	MIPS24K		25333162	12666581	6333290	3166645	1583322	791661
	DSR-HD		24960719	12480359	6240179	3120089	1560044	780022
djpeg	R3000	811968	26570335	13285167	6642583	3321291	1660645	830322
	MIPS24K		26965952	13482976	6741488	3370744	1685372	842686
	DSR-HD		26570335	13285167	6642583	3321291	1660645	830322
rgbcmy01	R3000	230824	7560845	3780422	1890211	945105	472552	236276
	MIPS24K		7682788	3841394	1920697	960348	480174	240087
	DSR-HD		7560845	3780422	1890211	945105	472552	236276
rgbhpg01	R3000	77224	2536518	1268259	634129	317064	158532	79266
	MIPS24K		2586127	1293063	646531	323265	161632	80816
	DSR-HD		2536518	1268259	634129	317064	158532	79266
rgbyiq01	R3000	230824	7560845	3780422	1890211	945105	472552	236276
	MIPS24K		7682788	3841394	1920697	960348	480174	240087
	DSR-HD		7560845	3780422	1890211	945105	472552	236276
ttsprk01	R3000	49844	1640905	820452	410226	205113	102556	51278
	MIPS24K		1677621	838810	419405	209702	104851	52425
	DSR-HD		1640905	820452	410226	205113	102556	51278
tblook01	R3000	12332	413870	206935	103467	51733	25866	12933
	MIPS24K		432921	216460	108230	54115	27057	13528
	DSR-HD		413870	206935	103467	51733	25866	12933
rspeed01	R3000	2084	78653	39326	19663	9831	4915	2457
	MIPS24K		92878	46439	23219	11609	5804	2902
	DSR-HD		78653	39326	19663	9831	4915	2457
puwmod01	R3000	9824	331832	165916	82958	41479	20739	10369
	MIPS24K		349702	174851	87425	43712	21856	10928
	DSR-HD		331832	165916	82958	41479	20739	10369
pntrch01	R3000	5516	190916	95458	47729	23864	11932	5966
	MIPS24K		206756	103378	51689	25844	12922	6461
	DSR-HD		190916	95458	47729	23864	11932	5966
canrdr01	R3000	6088	209626	104813	52406	26203	13101	6550
	MIPS24K		225736	112868	56434	28217	14108	7054
	DSR-HD		209626	104813	52406	26203	13101	6550
cacheb01	R3000	1008	43457	21728	10864	5432	2716	1358
	MIPS24K		57175	28587	14293	7146	3573	1786
	DSR-HD		43457	21728	10864	5432	2716	1358
bitmnp01	R3000	2776	101289	50644	25322	12661	6330	3165
	MIPS24K		115839	57919	28959	14479	7239	3619
	DSR-HD		101289	50644	25322	12661	6330	3165
aifirf01	R3000	4400	154411	77205	38602	19301	9650	4825
	MIPS24K		169726	84863	42431	21215	10607	5303
	DSR-HD		154411	77205	38602	19301	9650	4825
a2time01	R3000	2180	81793	40896	20448	10224	5112	2556
	MIPS24K		96063	48031	24015	12007	6003	3001
	DSR-HD		81793	40896	20448	10224	5112	2556

Table F.1: Data Memory Areas for EEMBC Applications: this table reports the silicon area of the primary memories required for the MIPS R3000, the MIPS24k and DSR-HD architectures. All memories were generated in a 130nm technology (more details in subsection 6.3): the surface to the next technology node was extrapolated by applying a multiplying factor of 0.5 as mentioned in [60].

Appendix G

HP Processor Performance: Conservative Wire Model

All the below graphs report the gain brought by the MIPS32 24K and DSR-HP processors over the MIPS R3000 on EEMBC benchmarks based on a pessimistic wire load model. The core clock frequency was set to 350MHz for the MIPS R3000 processor and 625MHz for the eight-stage architectures.

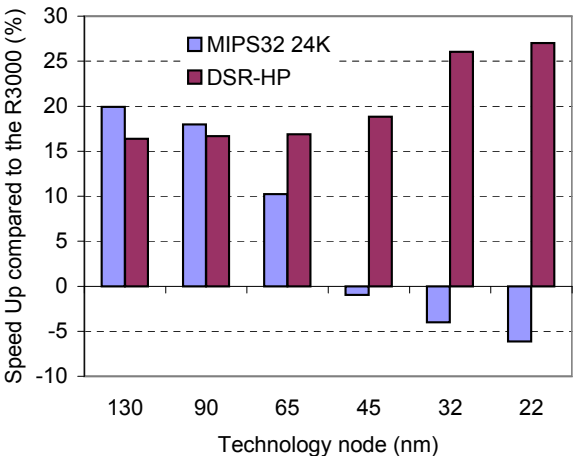

Figure G.1.1: Average Performance based on EEMBC Integer Benchmarks

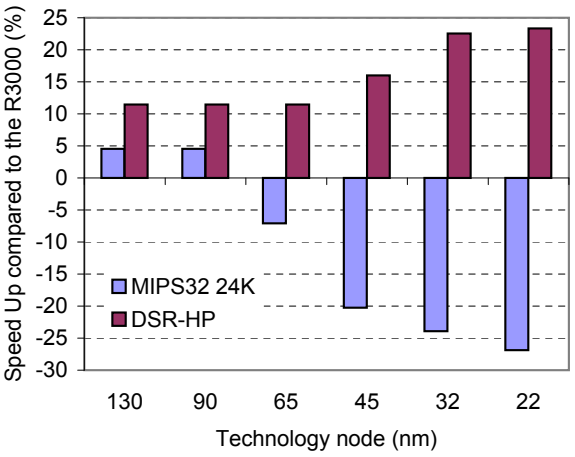

Figure G.1.2: ospf Benchmark Results

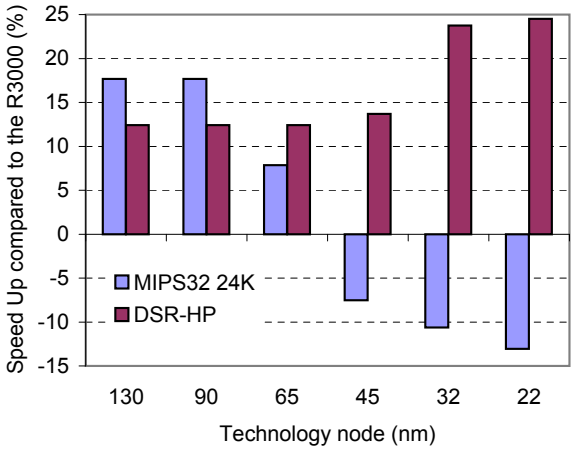

Figure G.1.3: pktflowb512k Benchmark Results

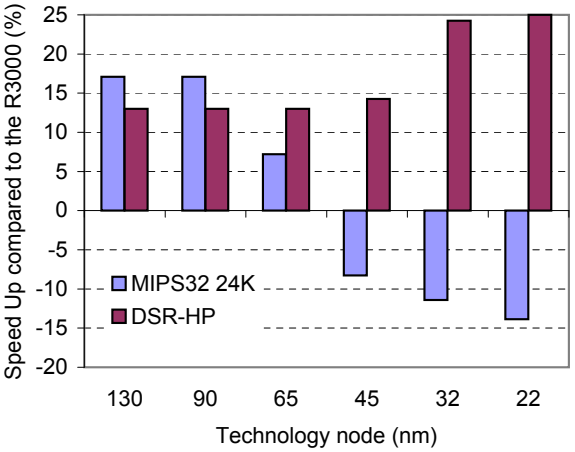

Figure G.1.4: pktflowb1m Benchmark Results

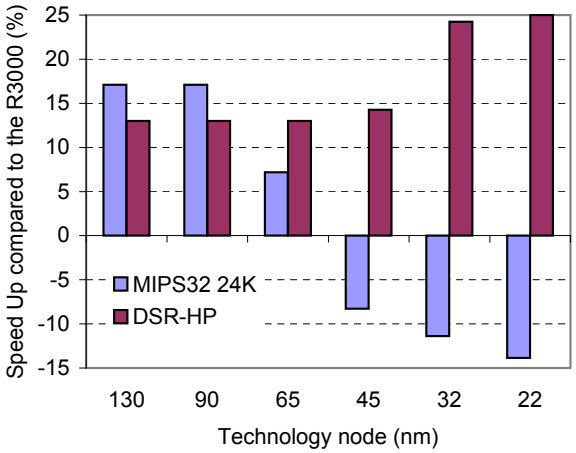

Figure G.1.5: pktflowb2m Benchmark Results

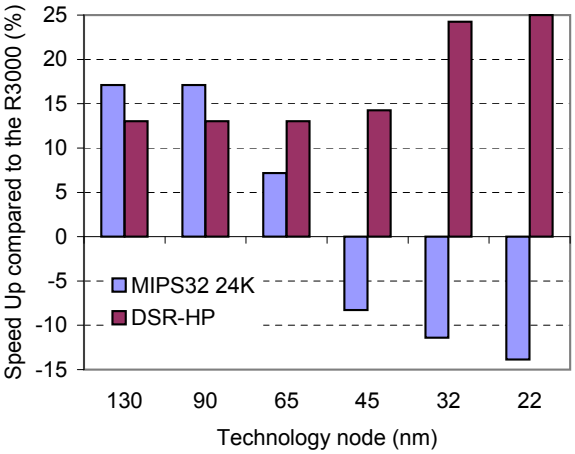

Figure G.1.6: pktflowb4m Benchmark Results

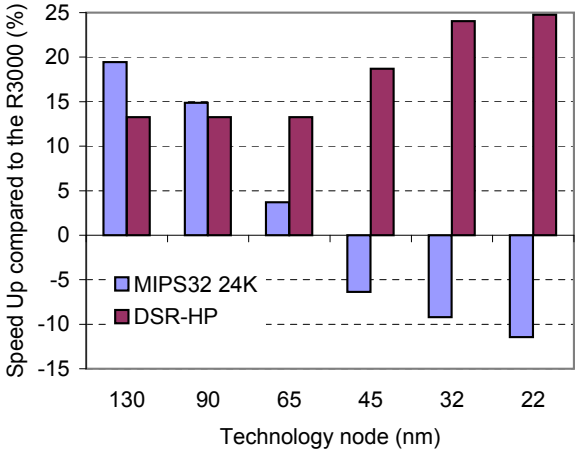

Figure G.1.7: routelookup Benchmark Results

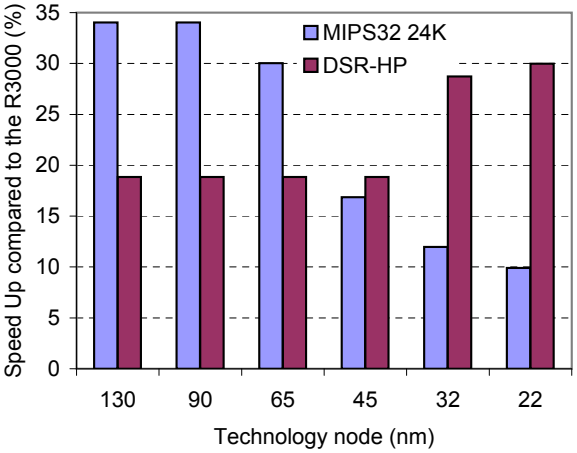

Figure G.1.8: autcor00data_1 Benchmark Results

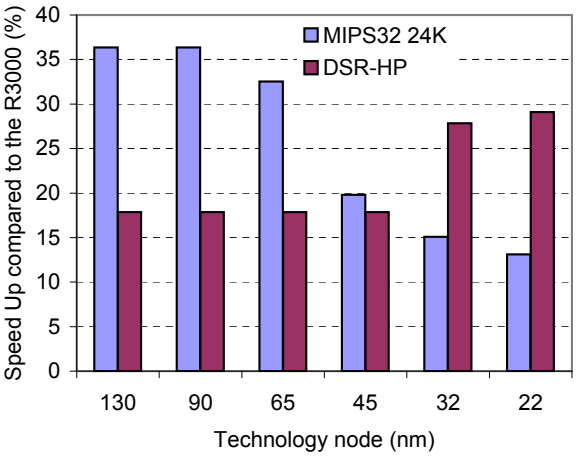

Figure G.1.9: autcor00data_2 Benchmark Results

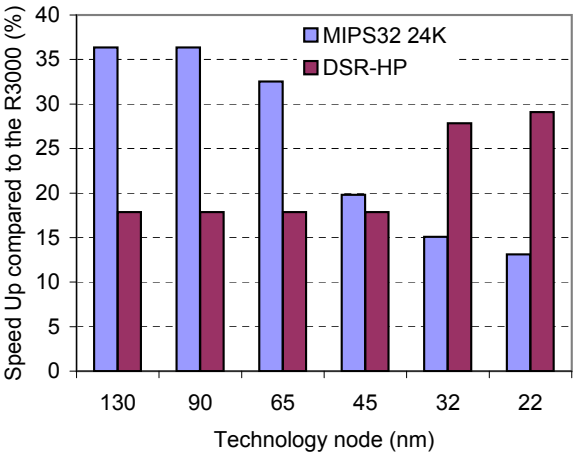

Figure G.1.10: autcor00data_3 Benchmark Results

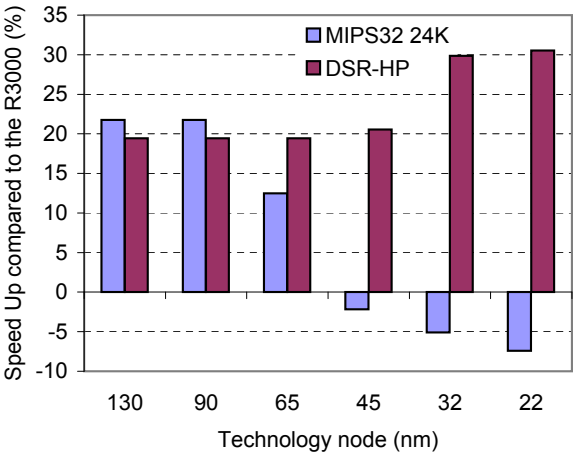

Figure G.1.11: conven00data_1 Benchmark Results

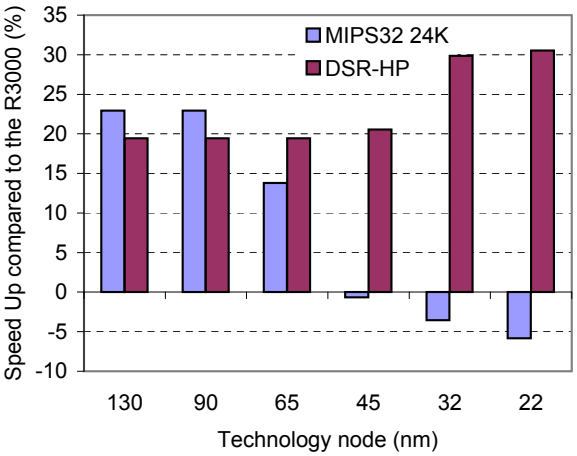

Figure G.1.12: conven00data_2 Benchmark Results

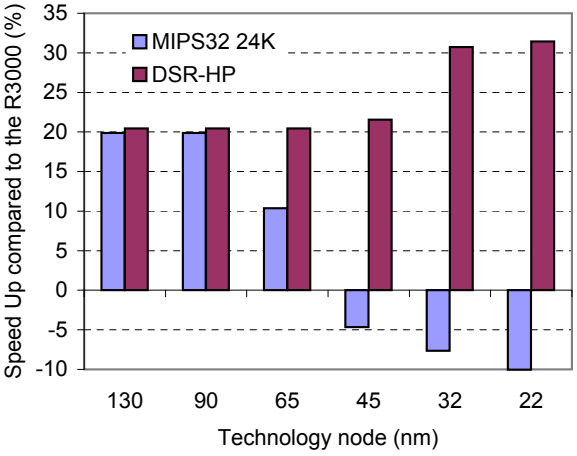

Figure G.1.13: conven00data_3 Benchmark Results

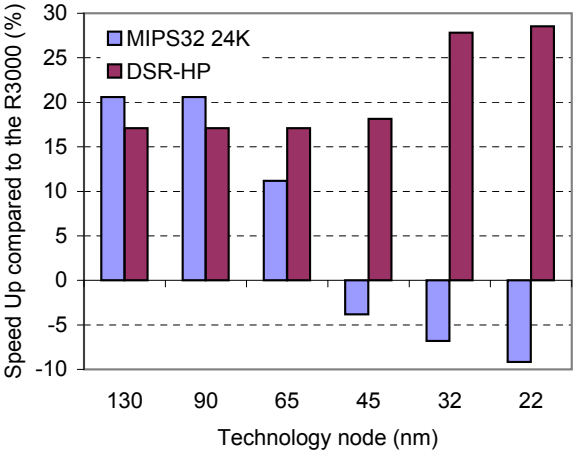

Figure G.1.14: fbital00data_2 Benchmark Results

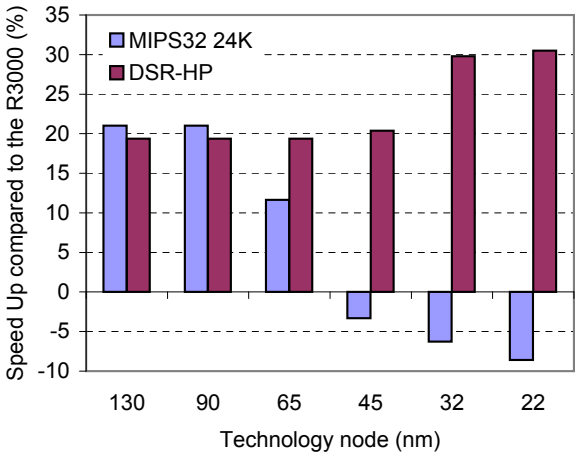

Figure G.1.15: fbital00data_3 Benchmark Results

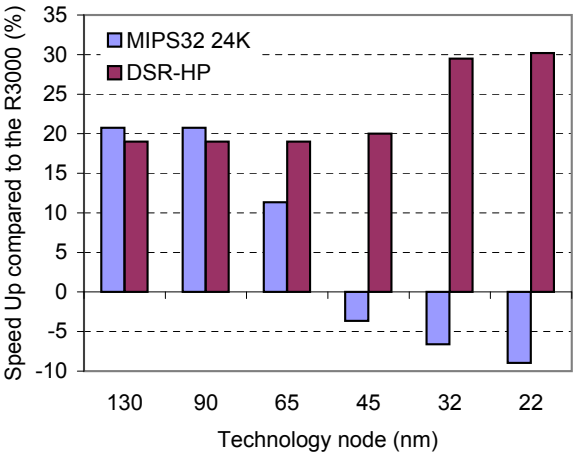

Figure G.1.16: fbital00data_6 Benchmark Results

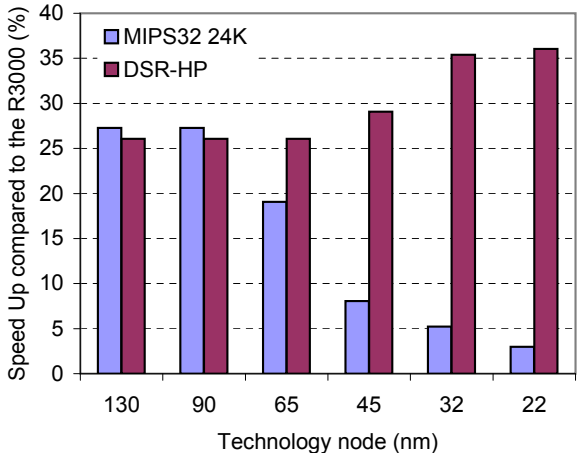

Figure G.1.17: fft00data_1 Benchmark Results

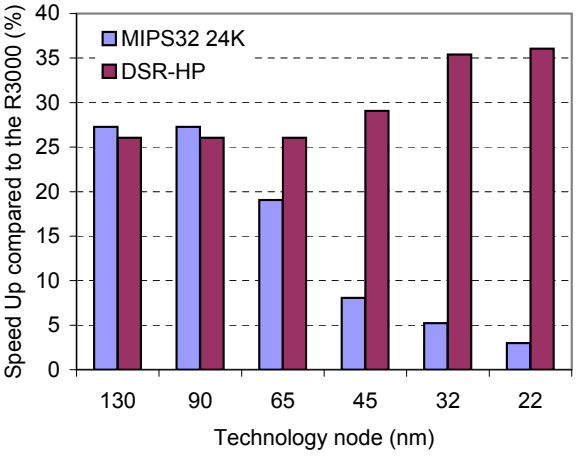

Figure G.1.18: fft00data_2 Benchmark Results

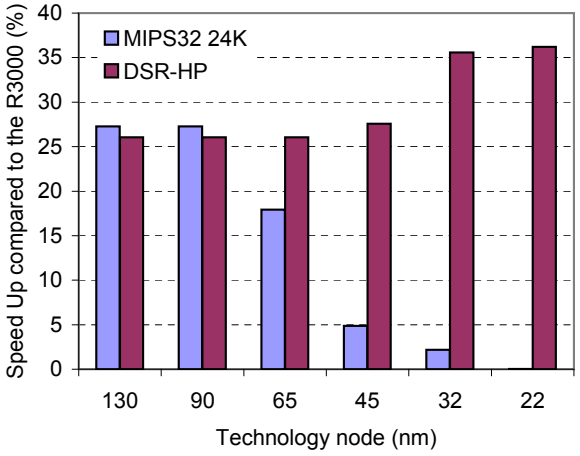

Figure G.1.19: fft00data_3 Benchmark Results

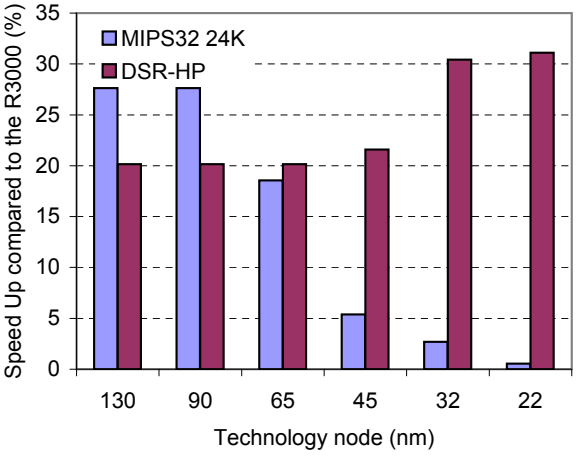

Figure G.1.20: viterb00data_1 Benchmark Results

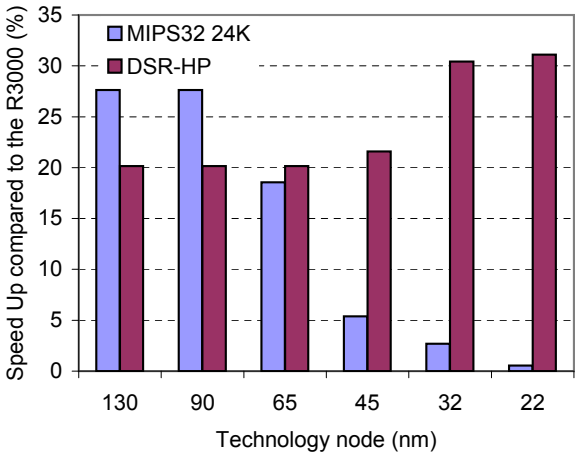

Figure G.1.21: viterb00data_2 Benchmark Results

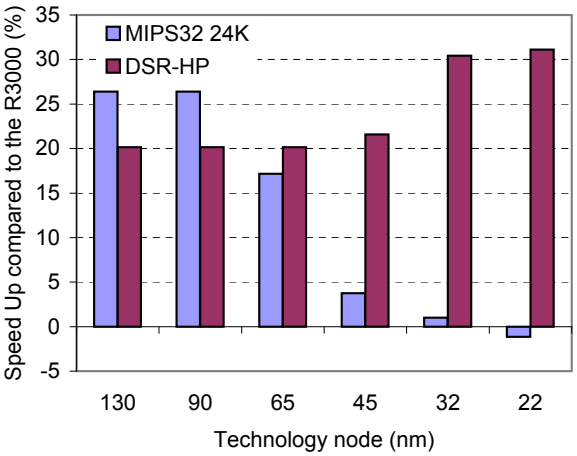

Figure G.1.22: viterb00data_3 Benchmark Results

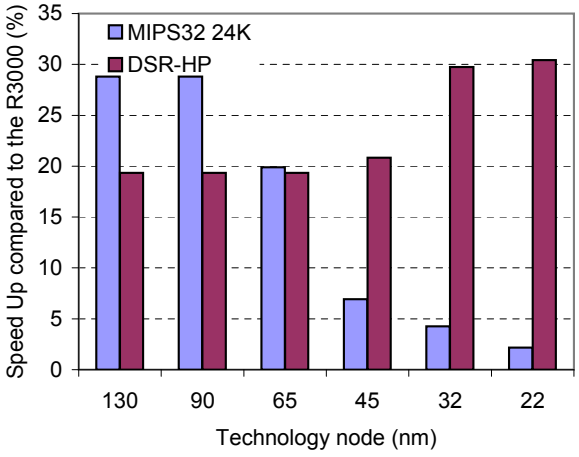

Figure G.1.23: viterb00data_4 Benchmark Results

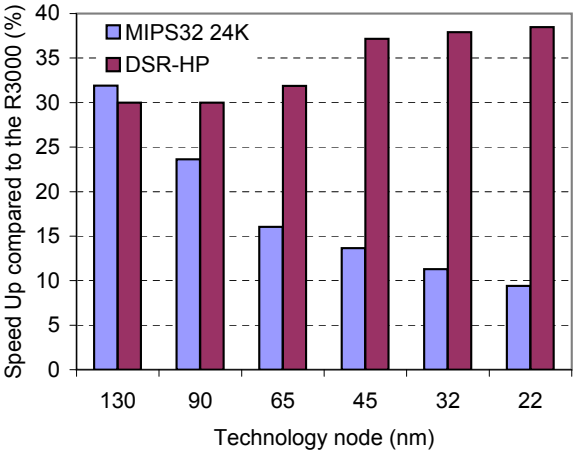

Figure G.1.24: bezier01fixed Benchmark Results

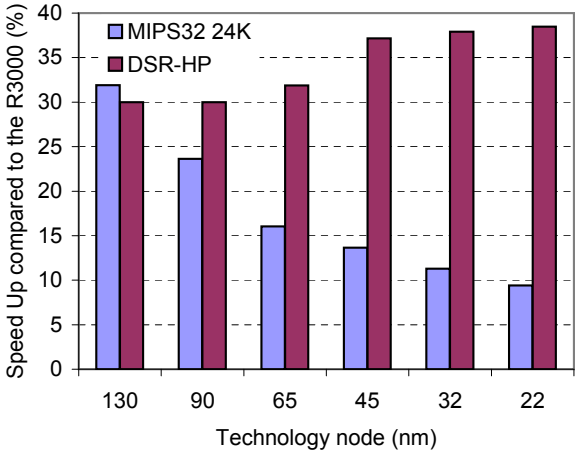

Figure G.1.25: bezier01float Benchmark Results

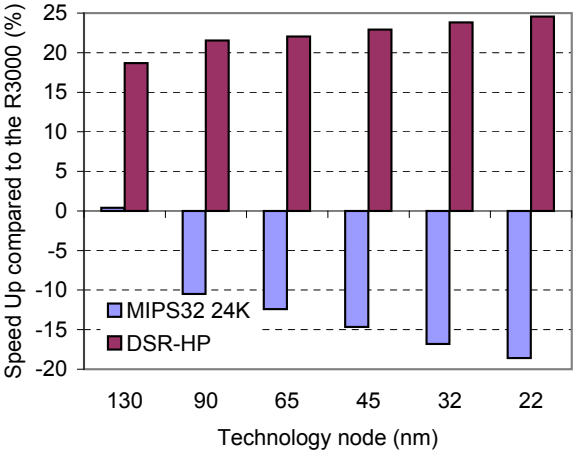

Figure G.1.26: dither01 Benchmark Results

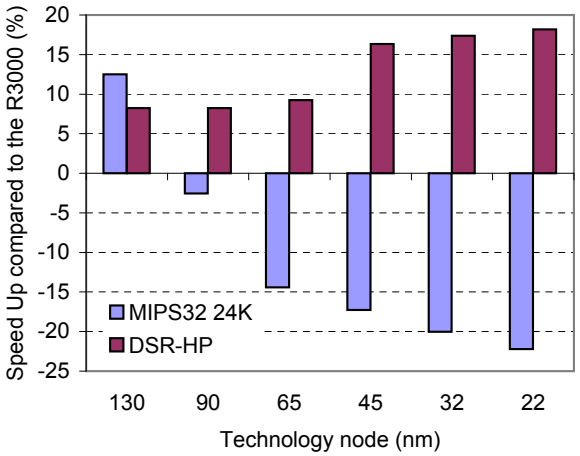

Figure G.1.27: rotate01 Benchmark Results

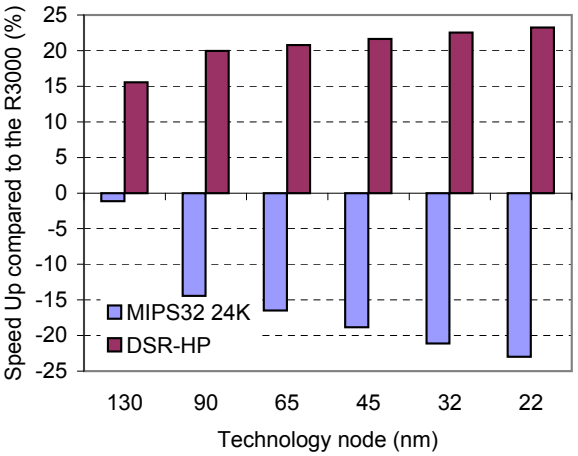

Figure G.1.28: text01 Benchmark Results

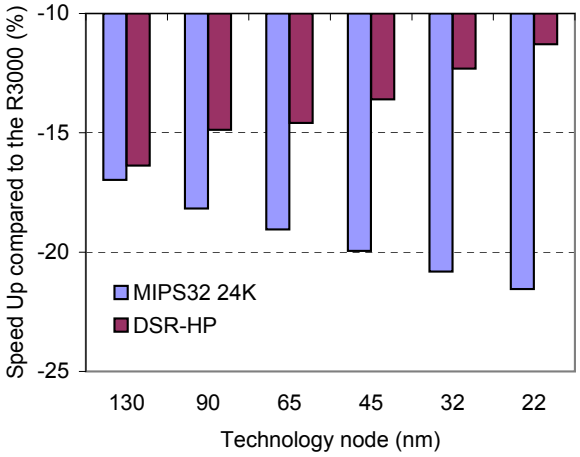

Figure G.1.29: djpeg Benchmark Results

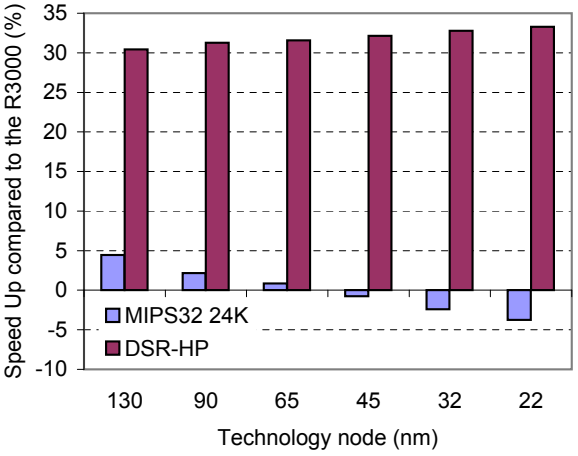

Figure G.1.30: rgbcmy01 Benchmark Results

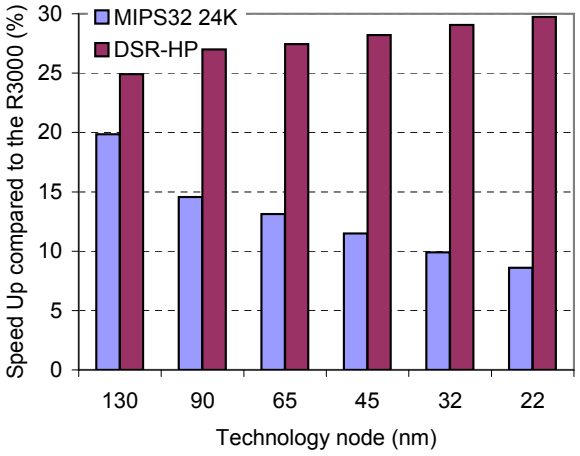

Figure G.1.31: rgbhpg01 Benchmark Results

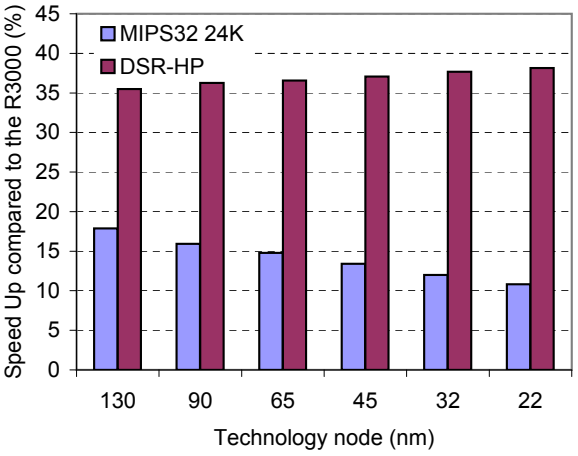

Figure G.1.32: rgbyiq01 Benchmark Results

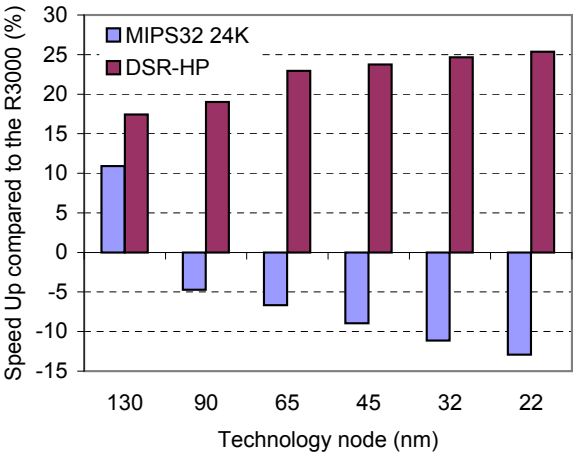

Figure G.1.33: ttsprk01 Benchmark Results

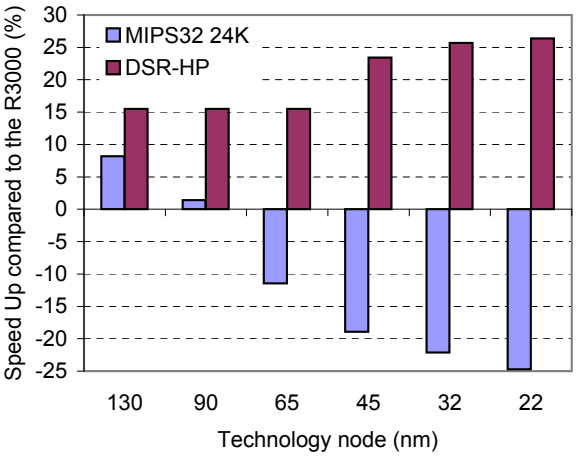

Figure G.1.34: tblook01 Benchmark Results

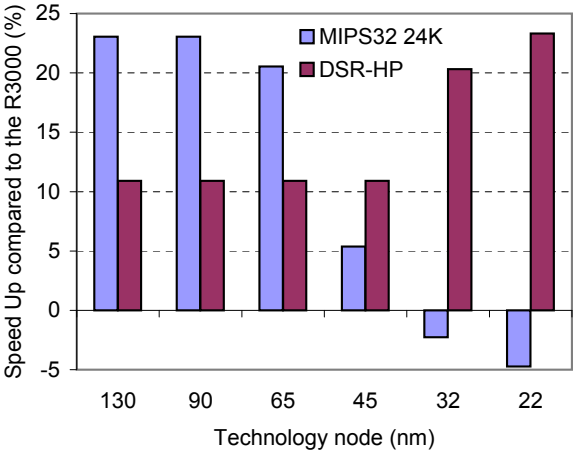

Figure G.1.35: rspeed01 Benchmark Results

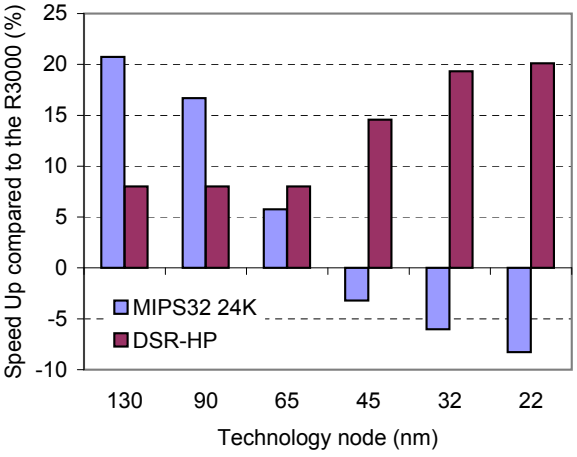

Figure G.1.36: puwmod01 Benchmark Results

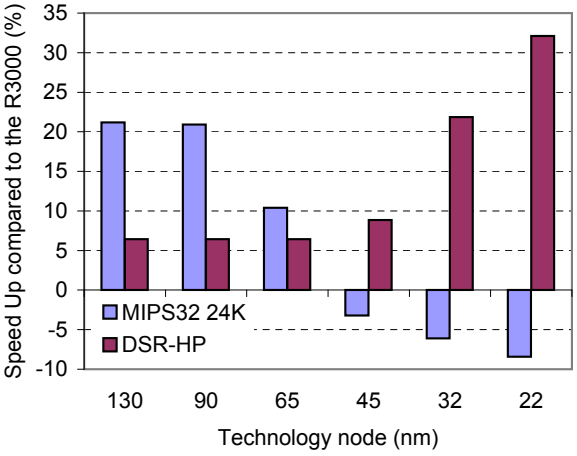

Figure G.1.37: pntrch01 Benchmark Results

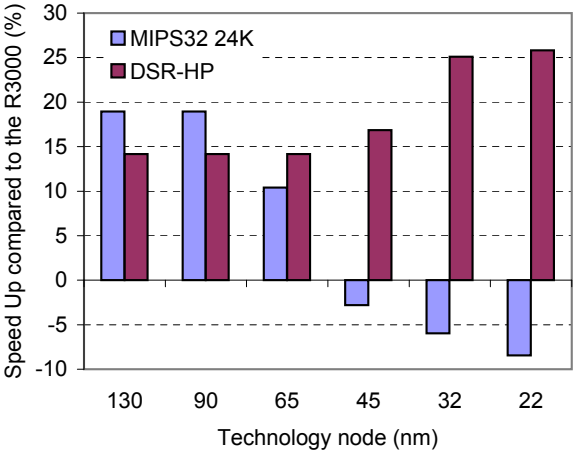

Figure G.1.38: canrdr01 Benchmark Results

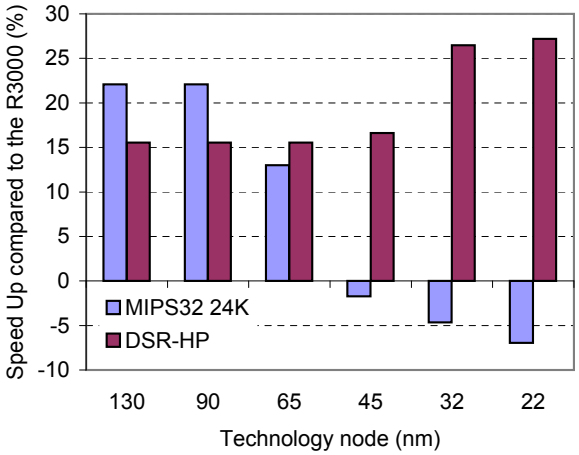

Figure G.1.39: cacheb01 Benchmark Results

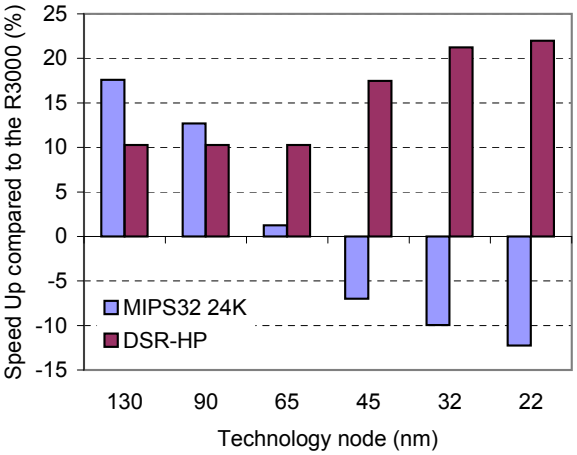

Figure G.1.40: bitmnp01 Benchmark Results

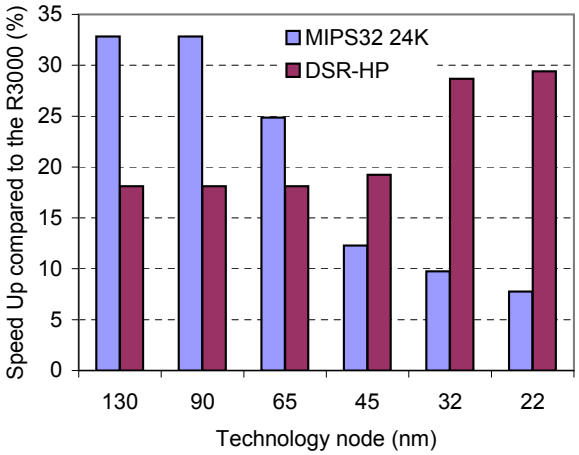

Figure G.1.41: aifirf01 Benchmark Results

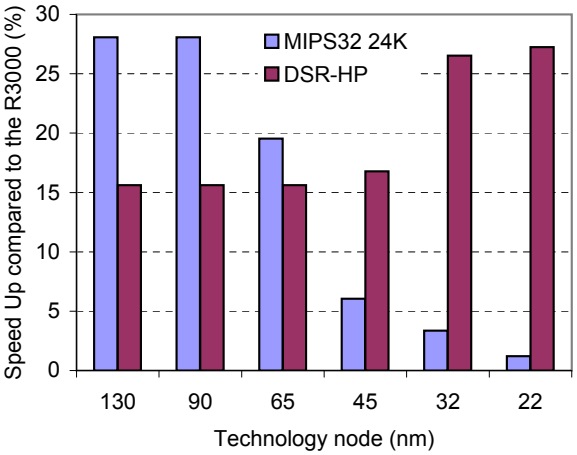

Figure G.1.42: a2time01 Benchmark Results

Appendix H

HP Processor Performance: Aggressive Wire Model

All the below graphs report the gain brought by the MIPS32 24K and DSR-HP processors over the MIPS R3000 on EEMBC benchmarks based on an optimistic wire load model. The core clock frequency was set to 350MHz for the MIPS R3000 processor and 625MHz for the eight-stage architectures.

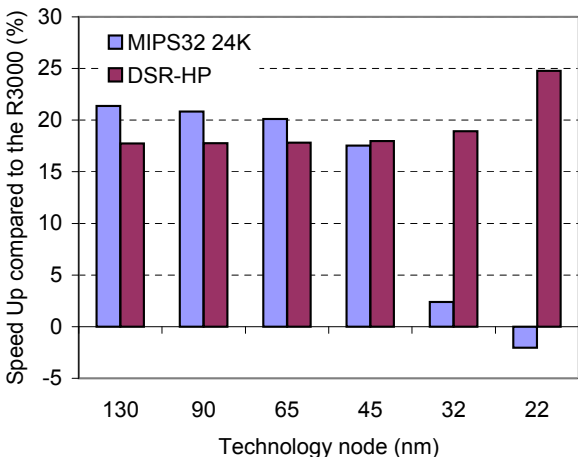

Figure H.1.43: Average Performance of EEMBC Integer Benchmarks

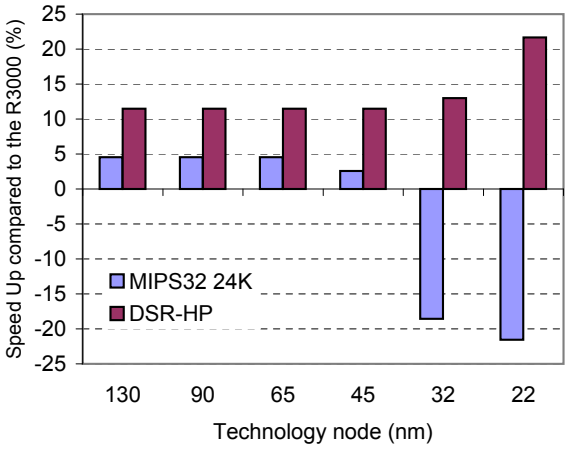

Figure H.1.44: ospf Benchmark Results

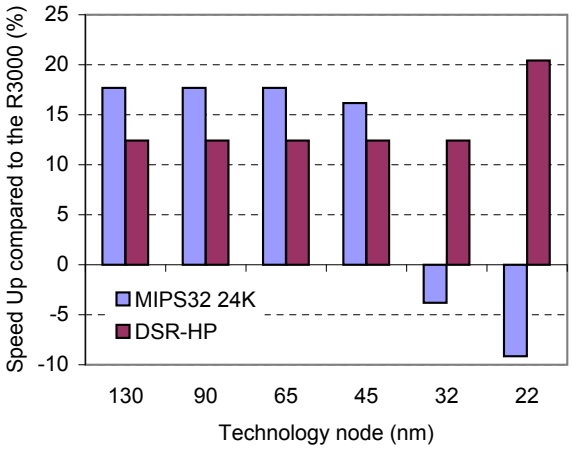

Figure H.1.45: pktflowb512k Benchmark Results

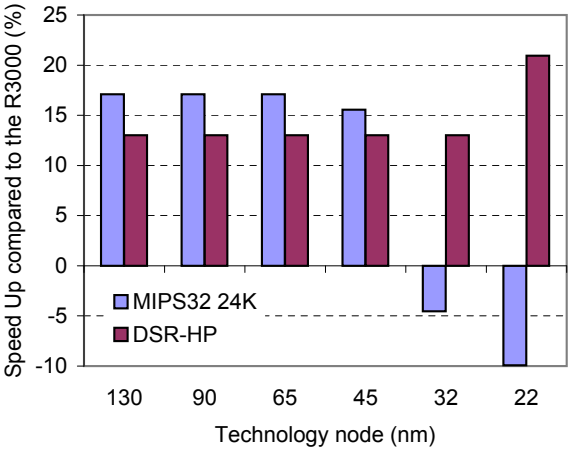

Figure H.1.46: pktflowb1m Benchmark Results

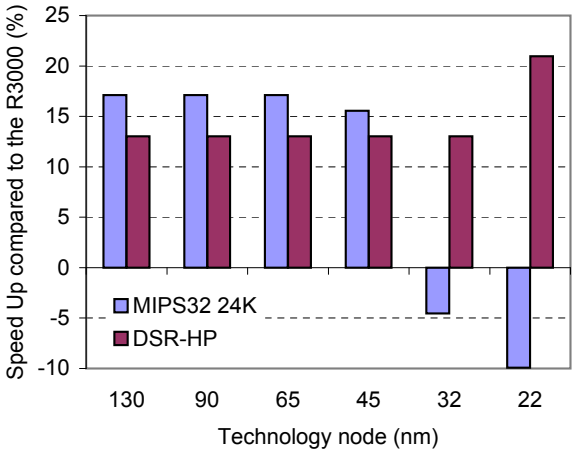

Figure H.1.47: pktflowb2m Benchmark Results

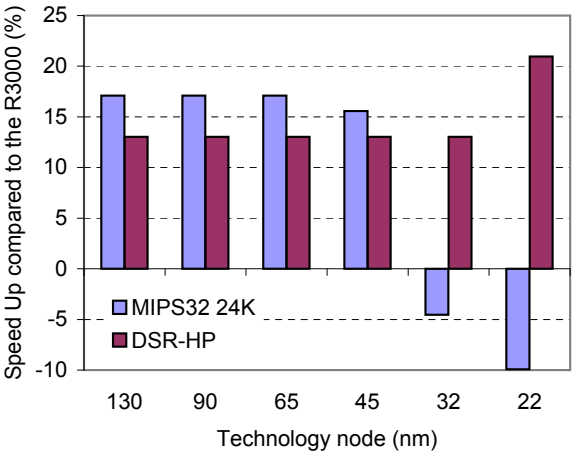

Figure H.1.48: pktflowb4m Benchmark Results

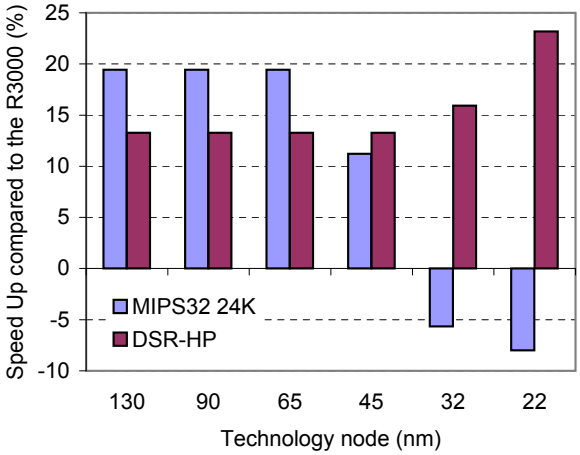

Figure H.1.49: routelookup Benchmark Results

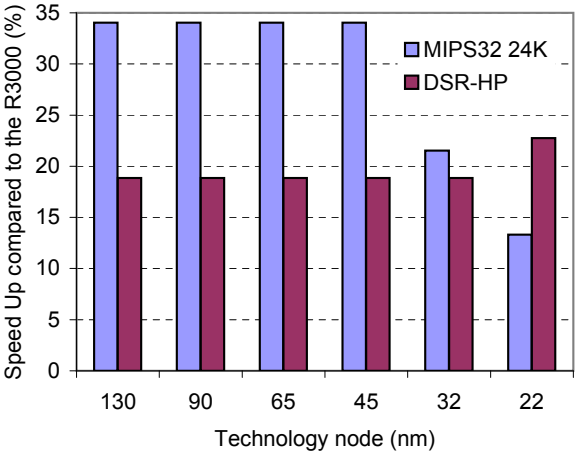

Figure H.1.50: autcor00data_1 Benchmark Results

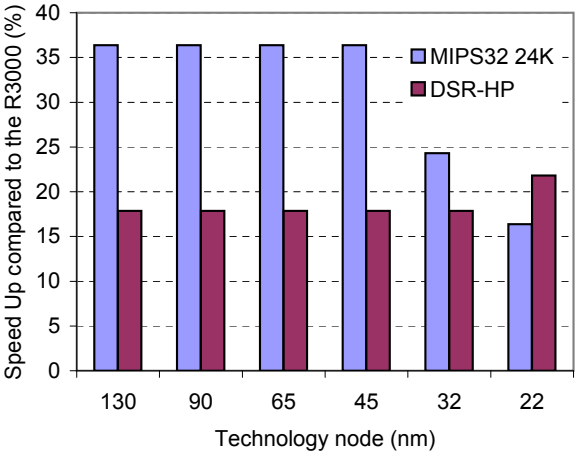

Figure H.1.51: autcor00data_2 Benchmark Results

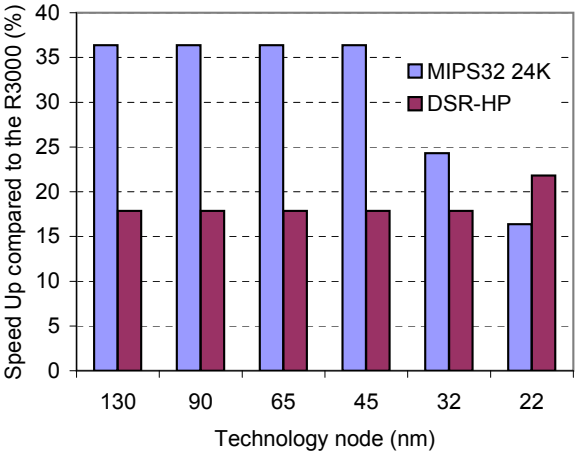

Figure H.1.52: autcor00data_3 Benchmark Results

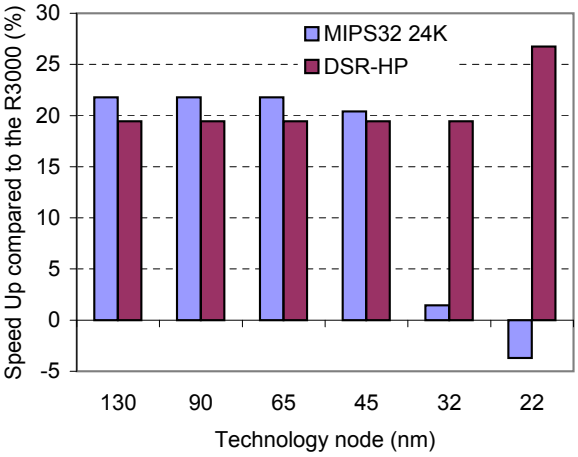

Figure H.1.53: conven00data_1 Benchmark Results

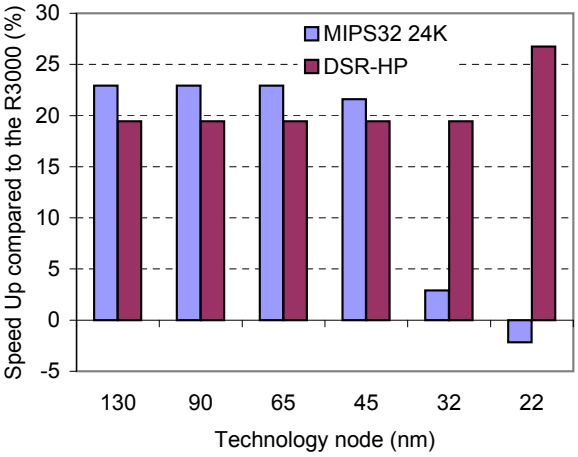

Figure H.1.54: conven00data_2 Benchmark Results

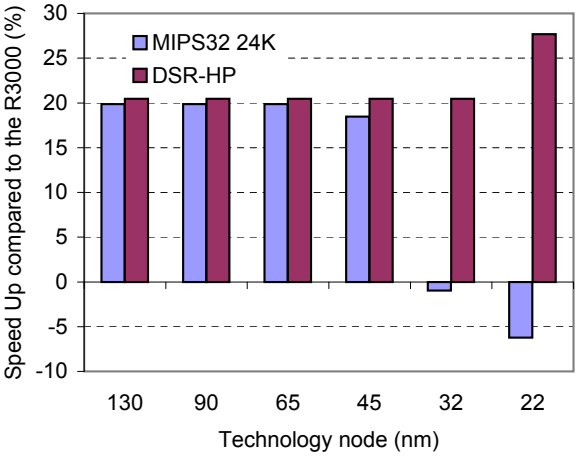

Figure H.1.55: conven00data_3 Benchmark Results

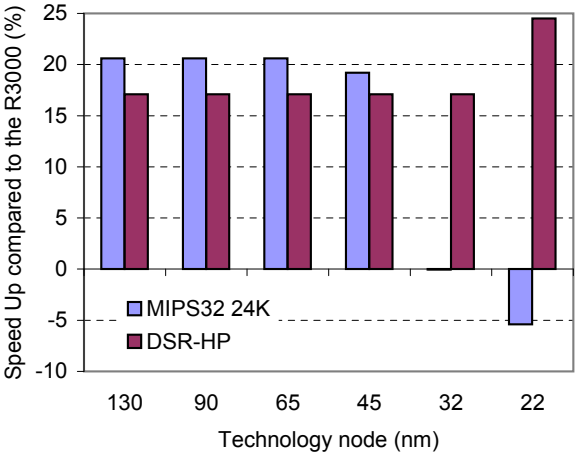

Figure H.1.56: fbital00data_2 Benchmark Results

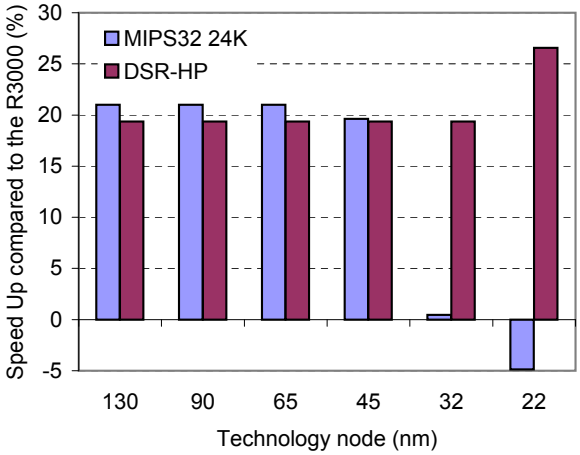

Figure H.1.57: fbital00data_3 Benchmark Results

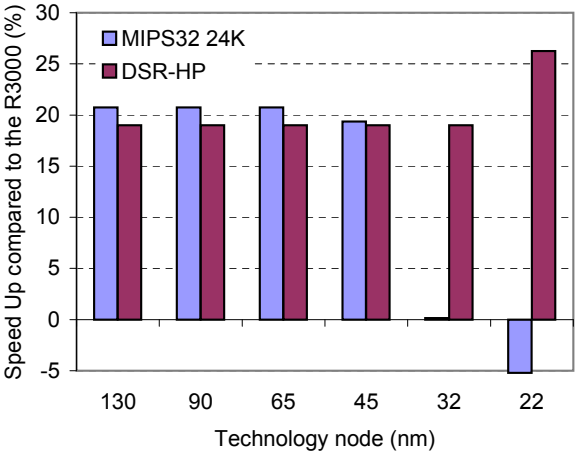

Figure H.1.58: fbital00data_6 Benchmark Results

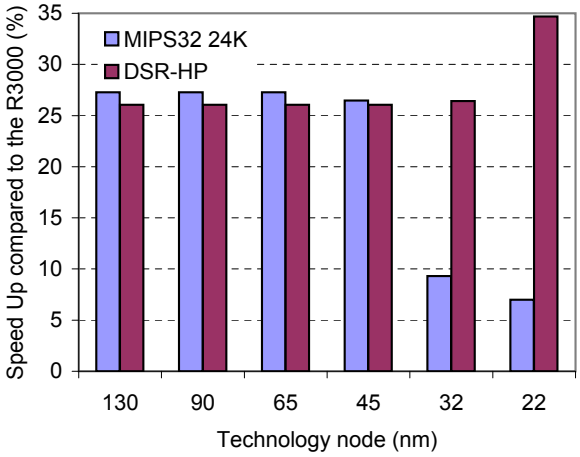

Figure H.1.59: fft00data_1 Benchmark Results

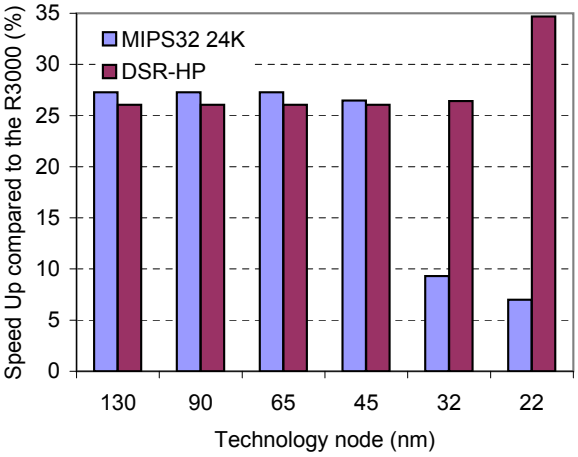

Figure H.1.60: fft00data_2 Benchmark Results

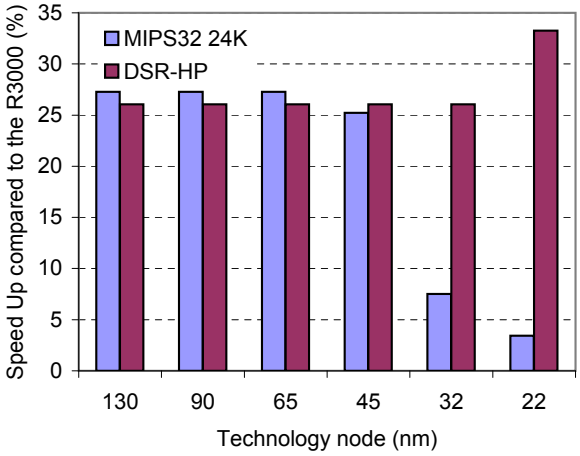

Figure H.1.61: fft00data_3 Benchmark Results

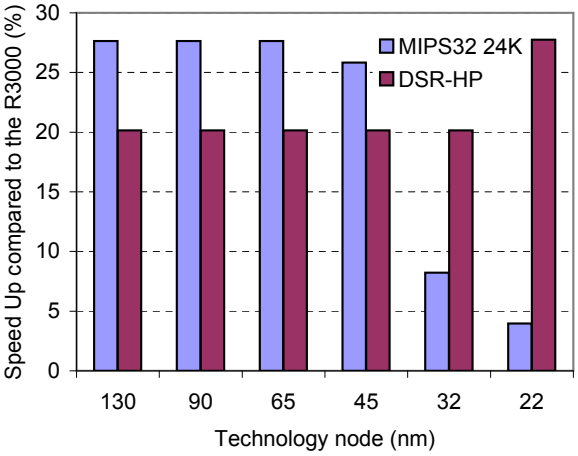

Figure H.1.62: viterb00data_1 Benchmark Results

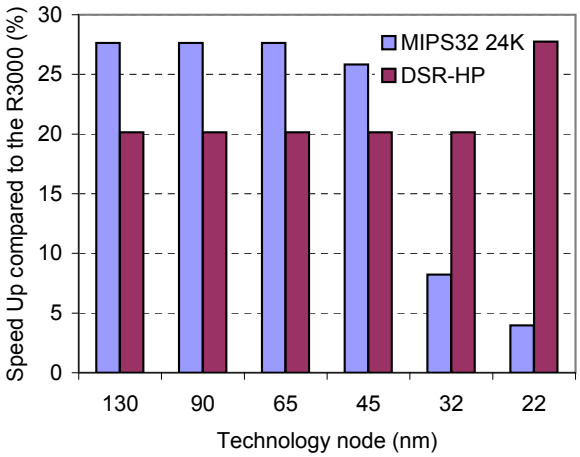

Figure H.1.63: viterb00data_2 Benchmark Results

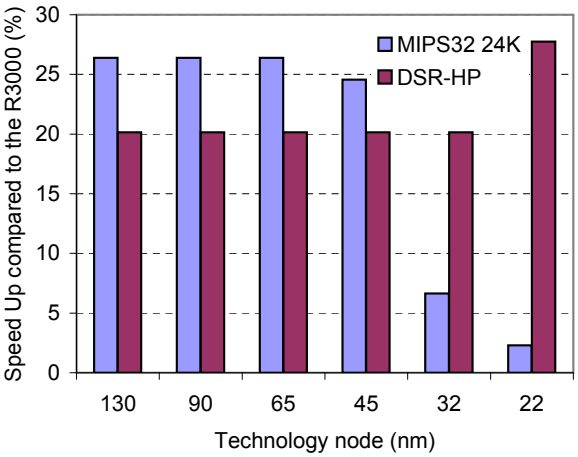

Figure H.1.64: viterb00data_3 Benchmark Results

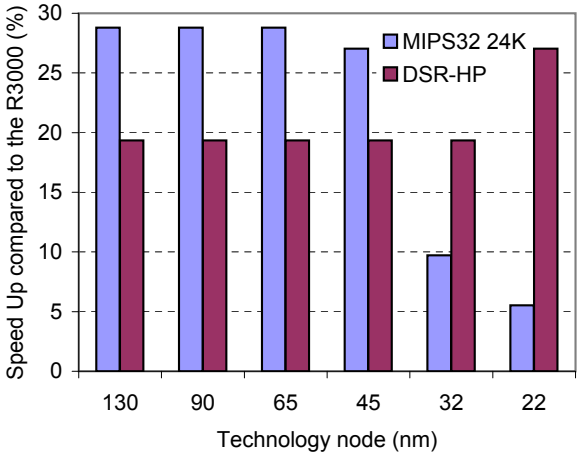

Figure H.1.65: viterb00data_4 Benchmark Results

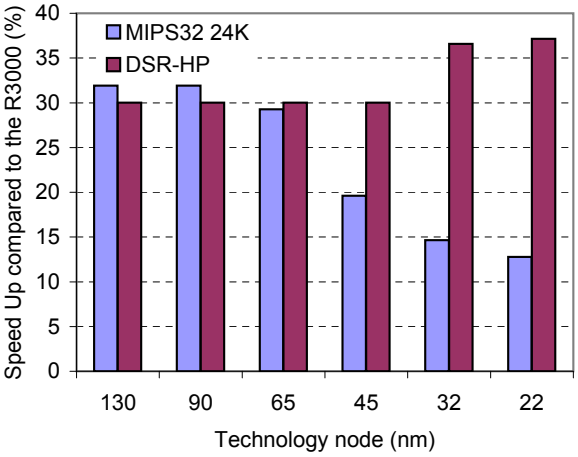

Figure H.1.66: bezier01fixed Benchmark Results

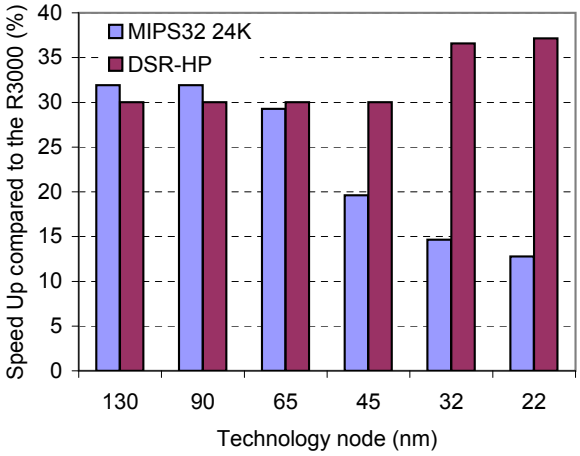

Figure H.1.67: bezier01float Benchmark Results

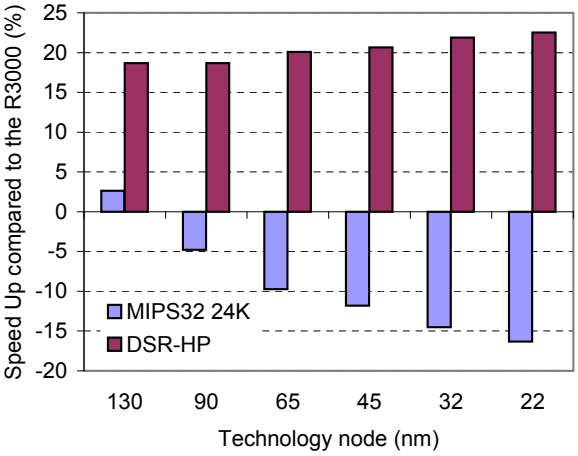

Figure H.1.68: dither01 Benchmark Results

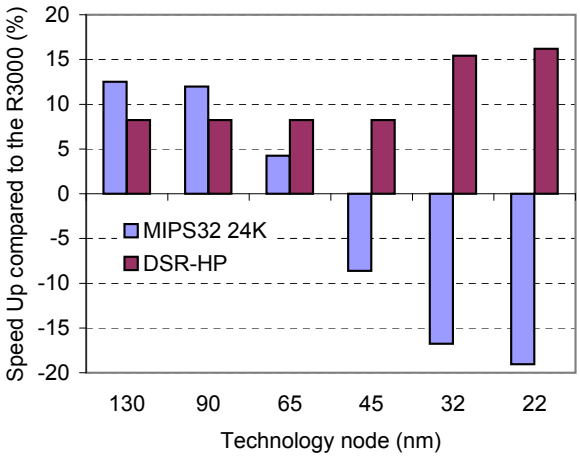

Figure H.1.69: rotate01 Benchmark Results

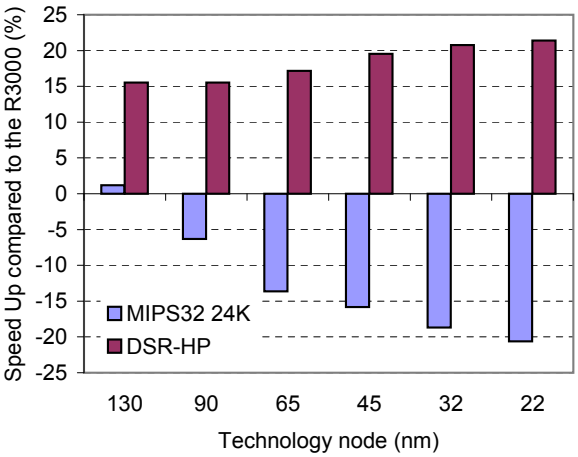

Figure H.1.70: text01 Benchmark Results

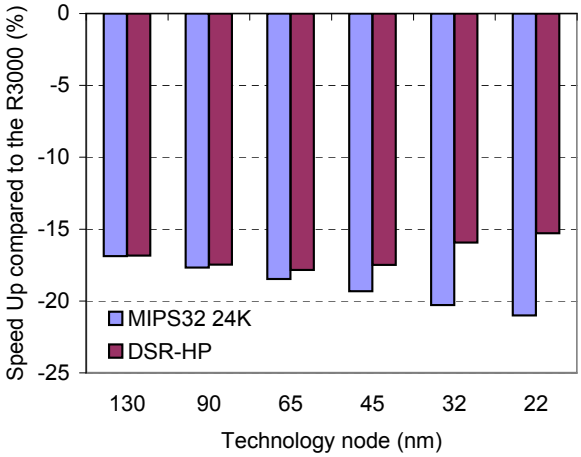

Figure H.1.71: djpeg Benchmark Results

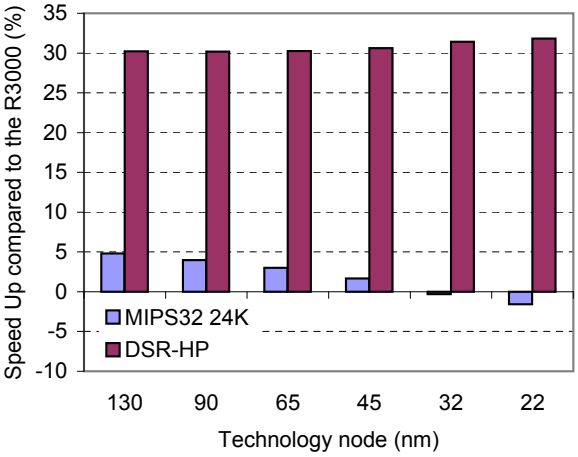

Figure H.1.72: rgbcmy01 Benchmark Results

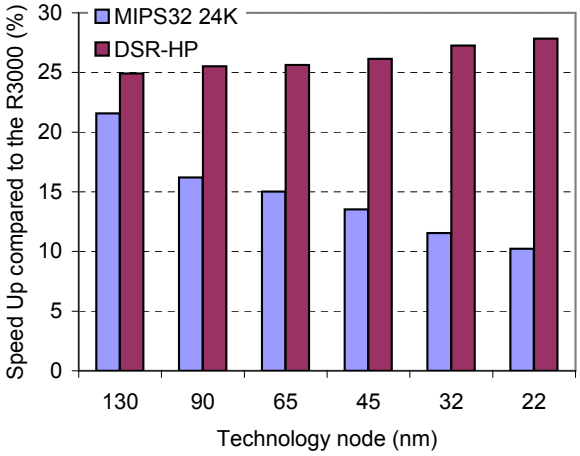

Figure H.1.73: rgbhpg01 Benchmark Results

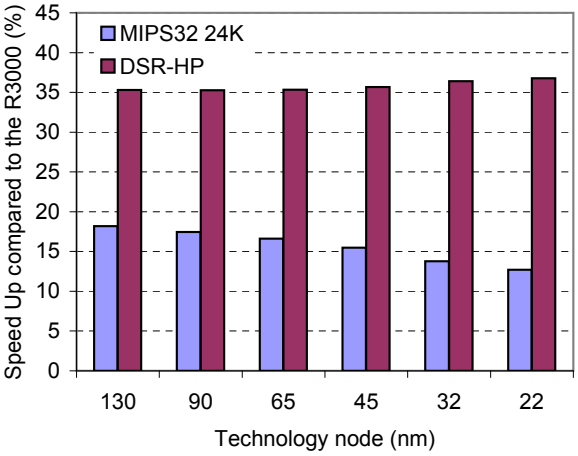

Figure H.1.74: rgbyiq01 Benchmark Results

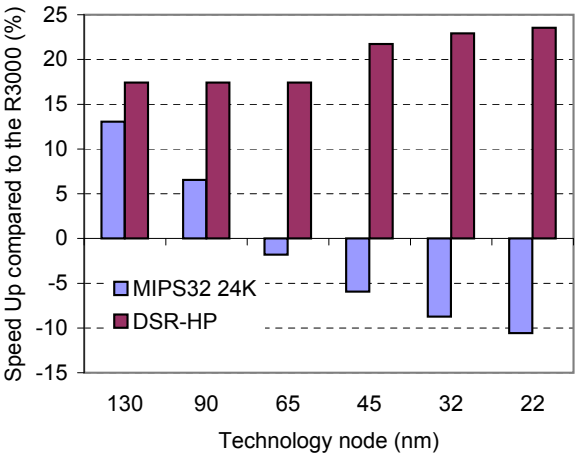

Figure H.1.75: ttsprk01 Benchmark Results

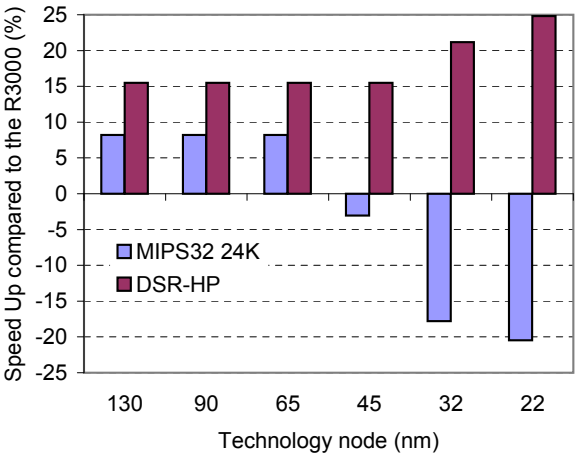

Figure H.1.76: tblook01 Benchmark Results

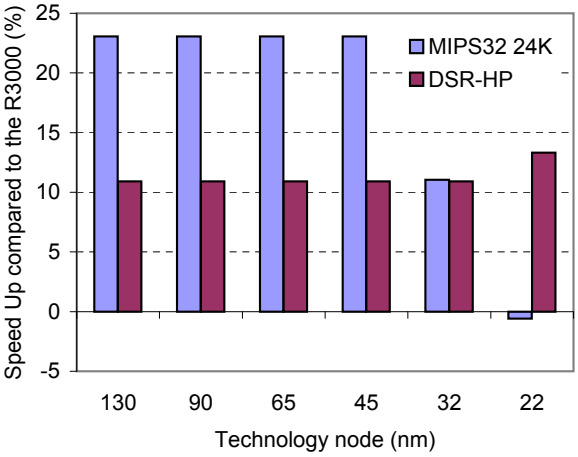

Figure H.1.77: rspeed01 Benchmark Results

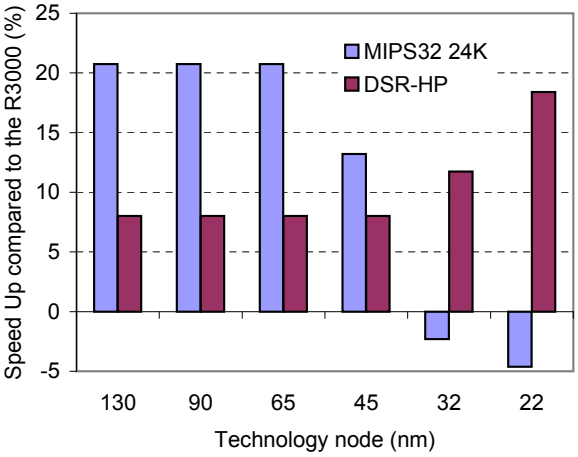

Figure H.1.78: puwmod01 Benchmark Results

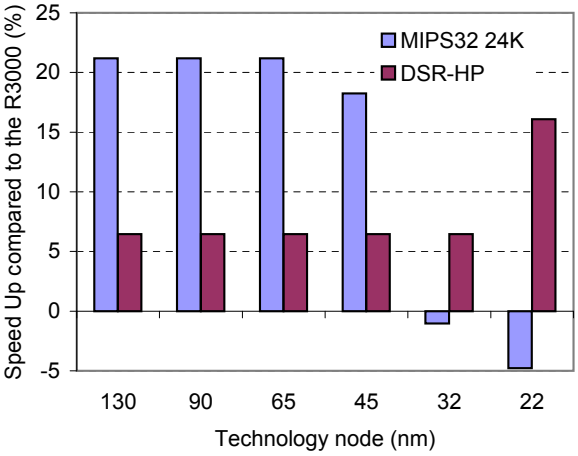

Figure H.1.79: pntrch01 Benchmark Results

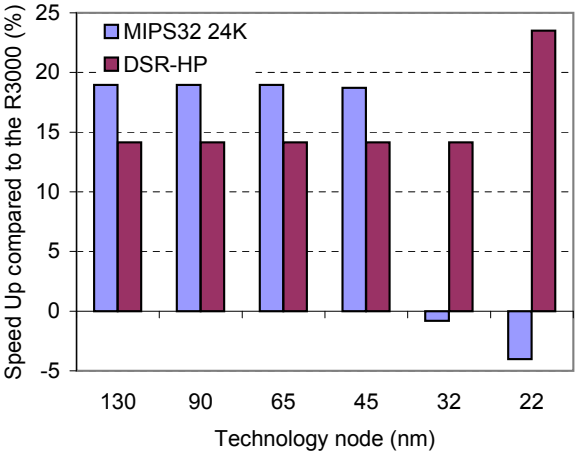

Figure H.1.80: canrdr01 Benchmark Results

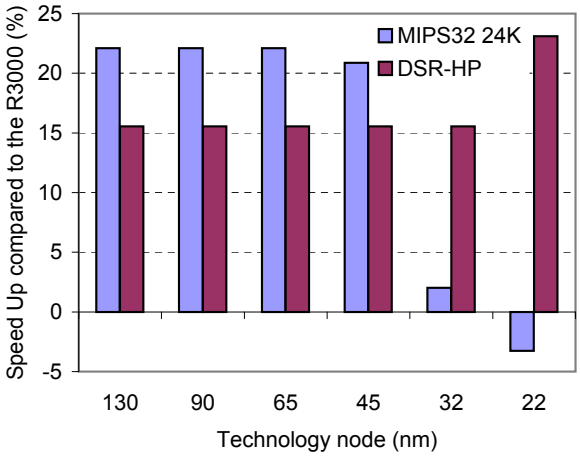

Figure H.1.81: cacheb01 Benchmark Results

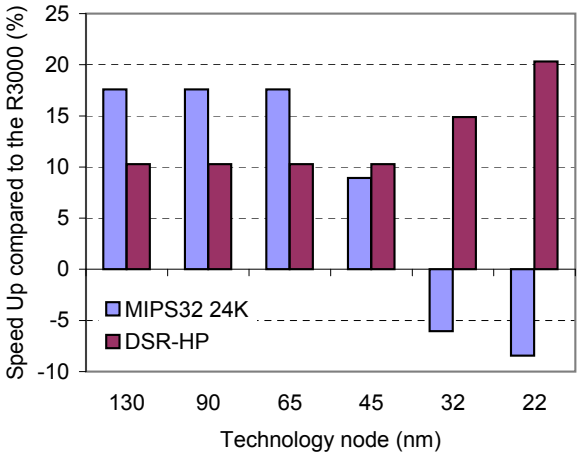

Figure H.1.82: bitmnp01 Benchmark Results

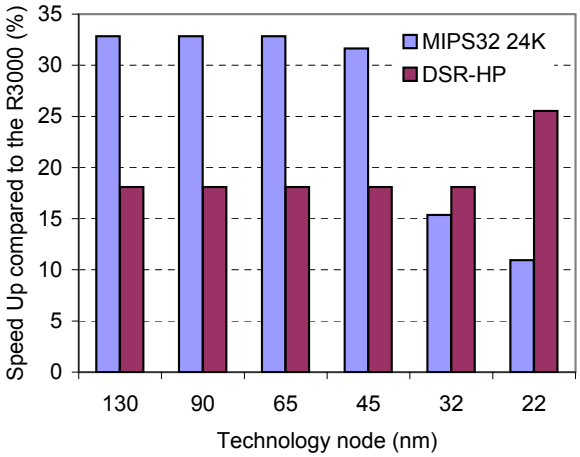

Figure H.1.83: aifirf01 Benchmark Results

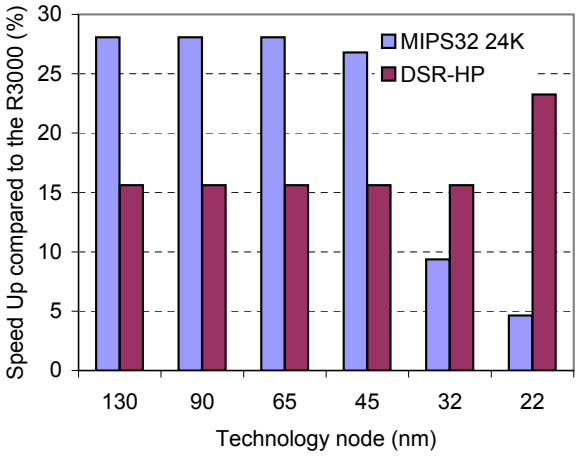

Figure H.1.84: a2time01 Benchmark Results

Appendix I

HD Processor Performance: Conservative Wire Model

All the below graphs report the gain brought by the MIPS32 24K and DSR-HD processors over the MIPS R3000 on EEMBC benchmarks based on a pessimistic wire load model. The core clock frequency was set to 350MHz for the MIPS R3000 processor and 625MHz for the eight-stage architectures.

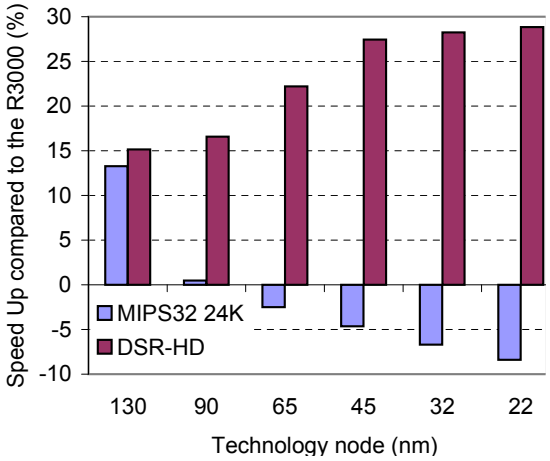

Figure I.1.85: Average Performance based on EEMBC Integer Benchmarks

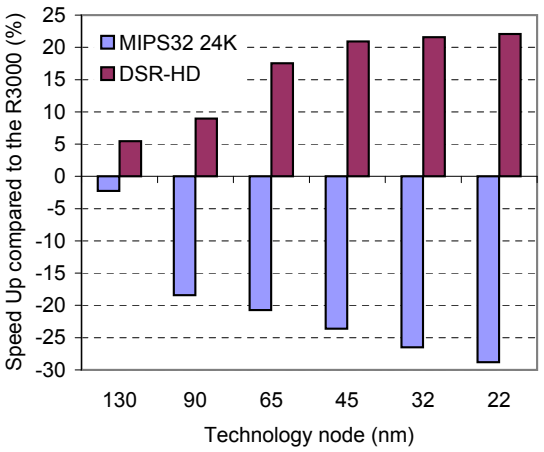

Figure I.1.86: ospf Benchmark Results

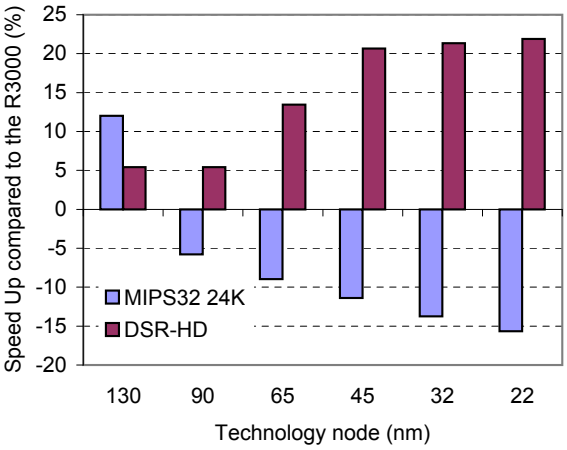

Figure I.1.87: pktflowb512k Benchmark Results

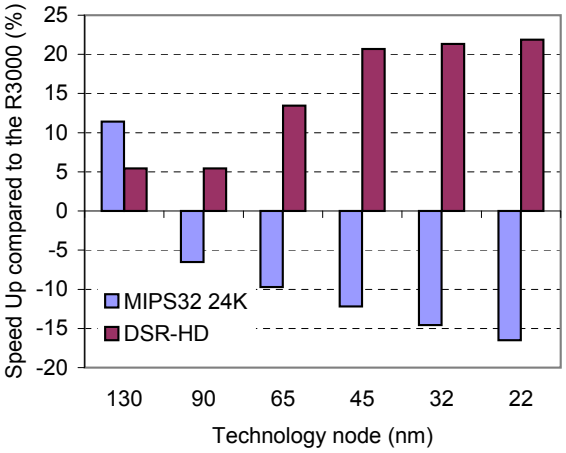

Figure I.1.88: pktflowb1m Benchmark Results

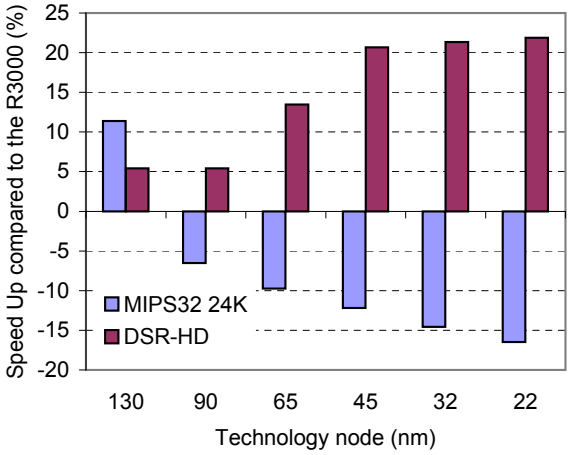

Figure I.1.89: pktflowb2m Benchmark Results

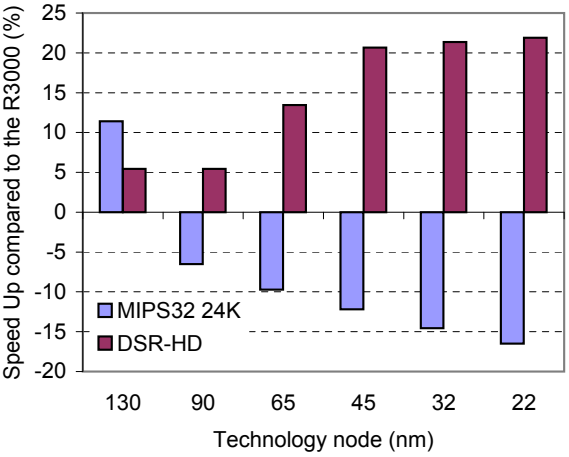

Figure I.1.90: pktflowb4m Benchmark Results

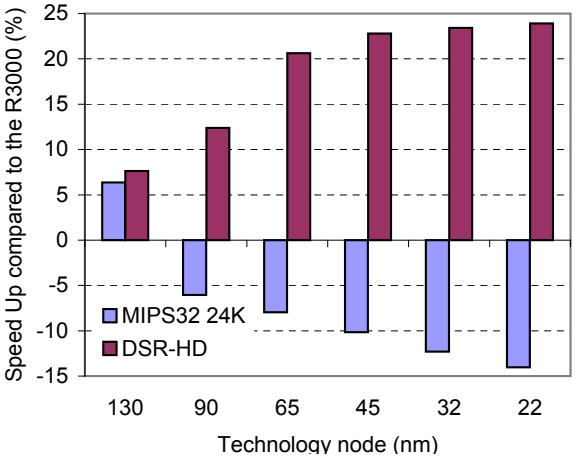

Figure I.1.91: routelookup Benchmark Results

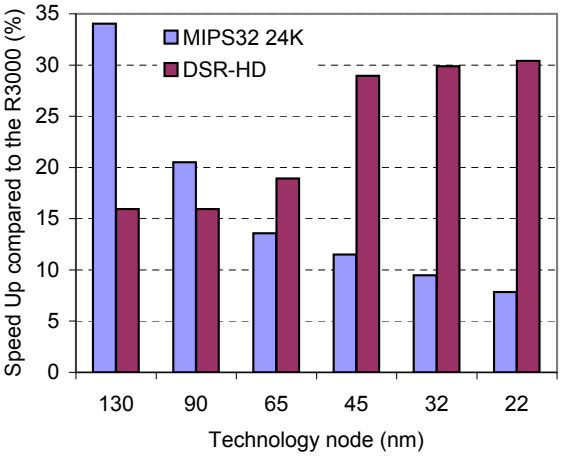

Figure I.1.92: autcor00data_1 Benchmark Results

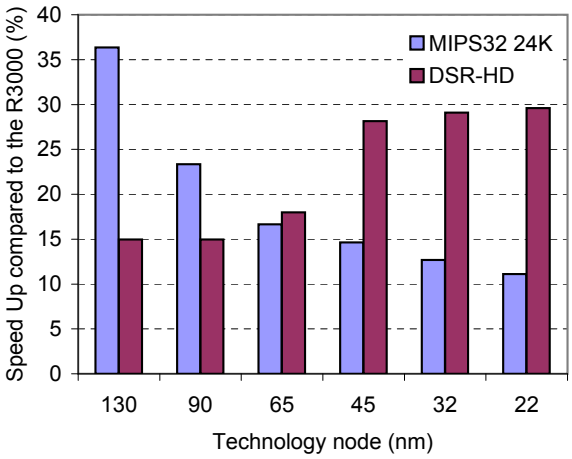

Figure I.1.93: autcor00data_2 Benchmark Results

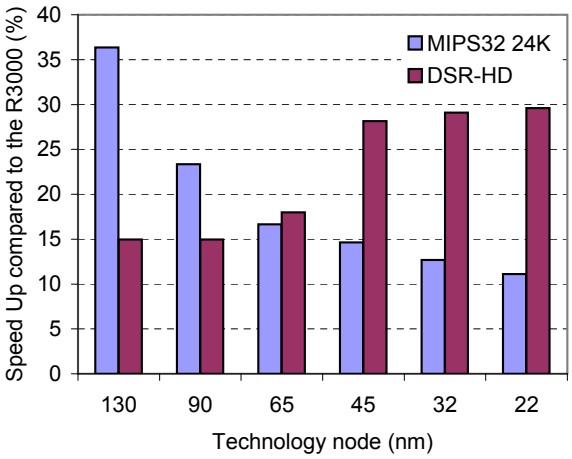

Figure I.1.94: autcor00data_3 Benchmark Results

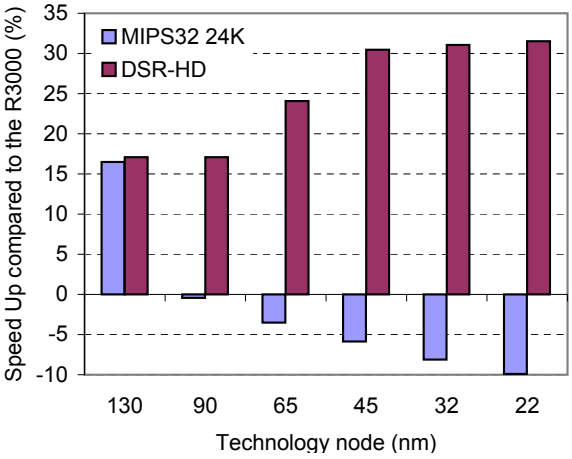

Figure I.1.95: conven00data_1 Benchmark Results

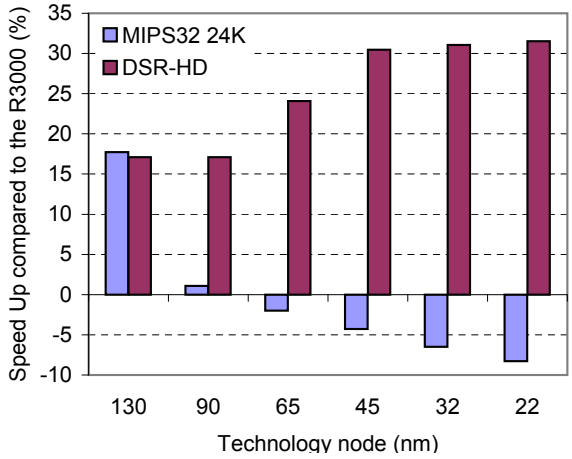

Figure I.1.96: conven00data_2 Benchmark Results

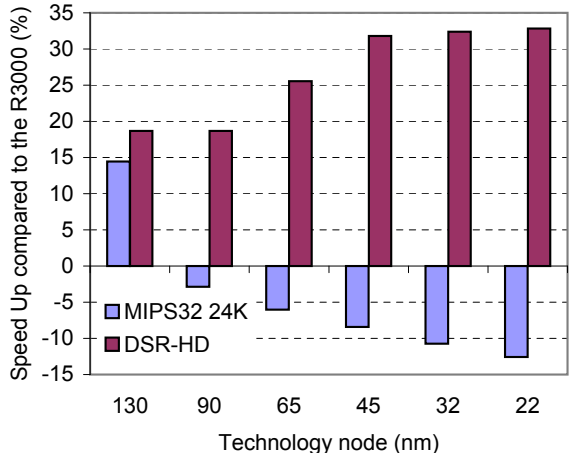

Figure I.1.97: conven00data_3 Benchmark Results

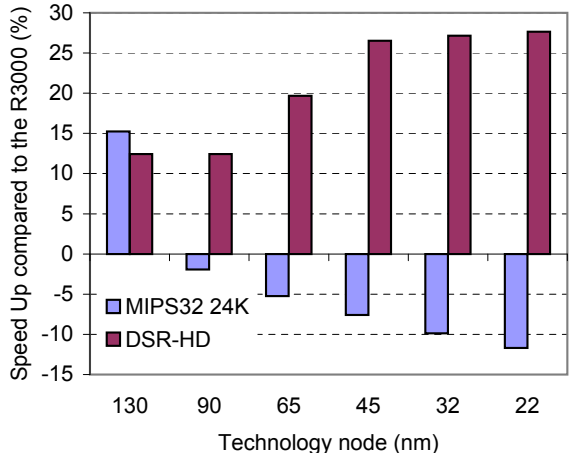

Figure I.1.98: fbital00data_2 Benchmark Results

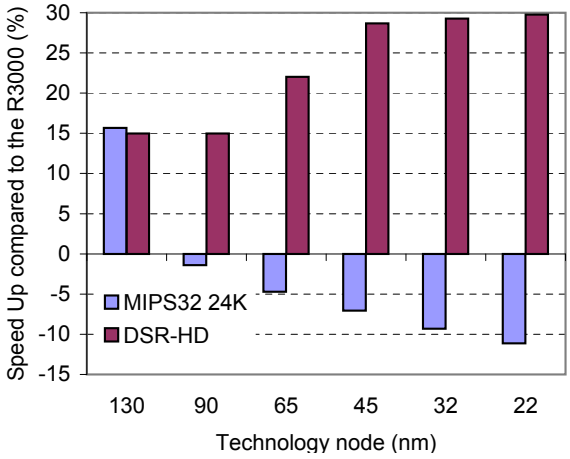

Figure I.1.99: fbital00data_3 Benchmark Results

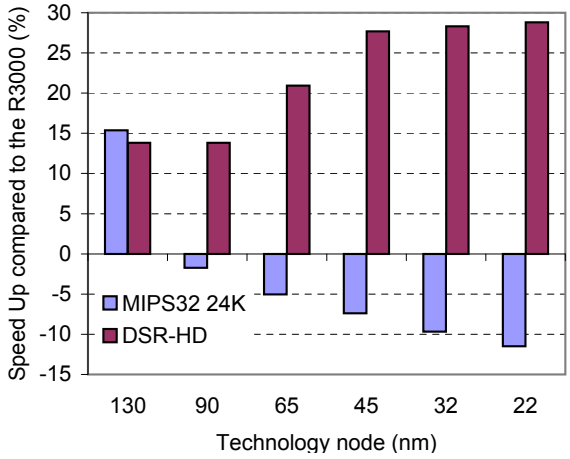

Figure I.1.100: fbital00data_6 Benchmark Results

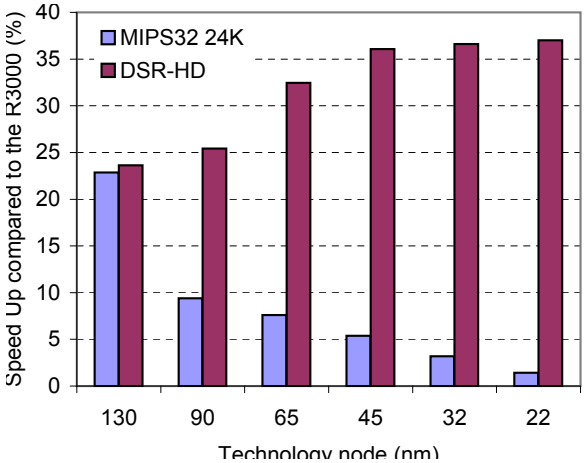

Figure I.1.101: fft00data_1 Benchmark Results

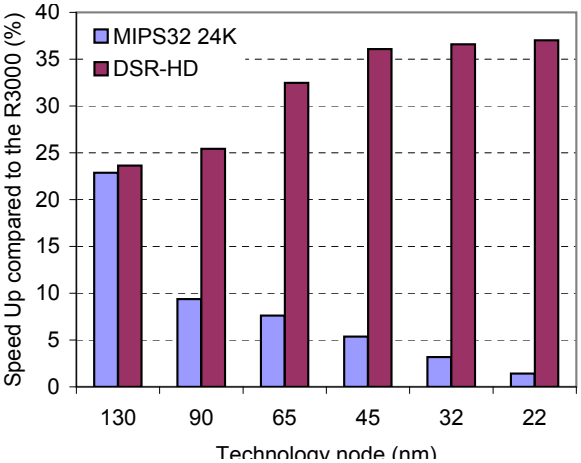

Figure I.1.102: fft00data_2 Benchmark Results

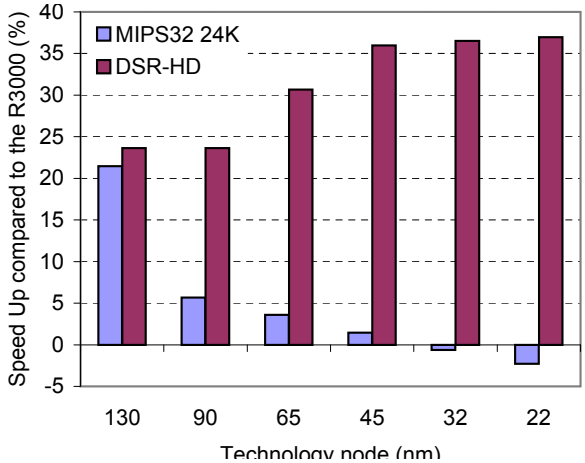

Figure I.1.103: fft00data_3 Benchmark Results

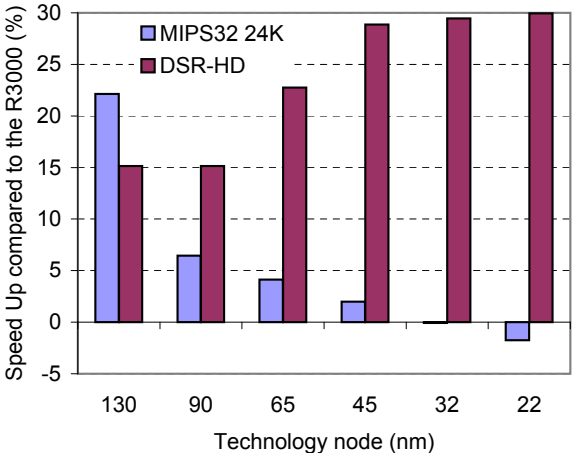

Figure I.1.104: viterb00data_1 Benchmark Results

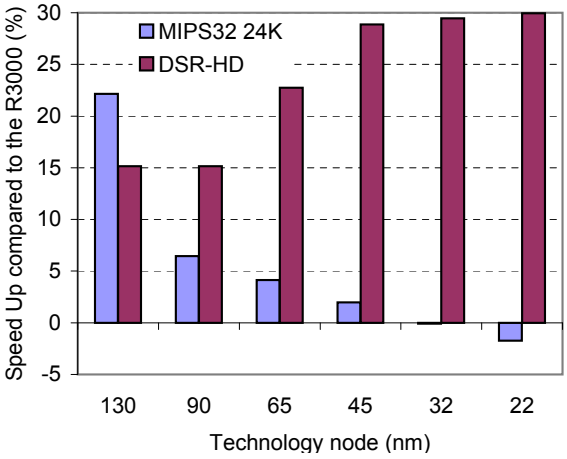

Figure I.1.105: viterb00data_2 Benchmark Results

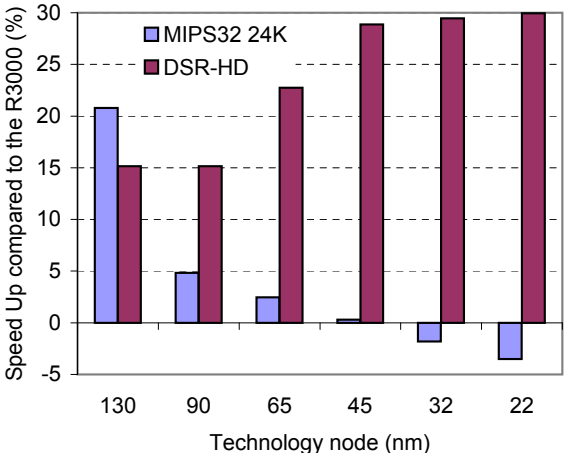

Figure I.1.106: viterb00data_3 Benchmark Results

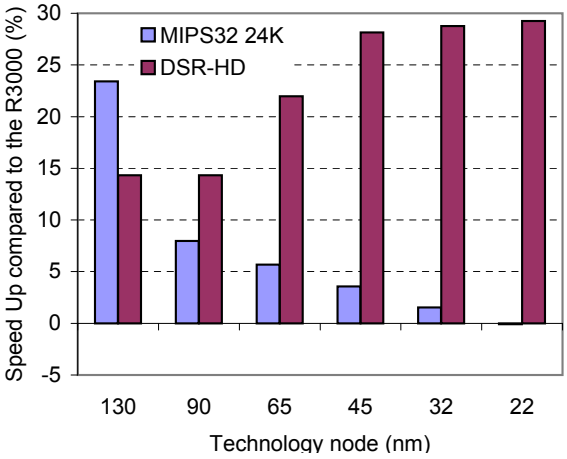

Figure I.1.107: viterb00data_4 Benchmark Results

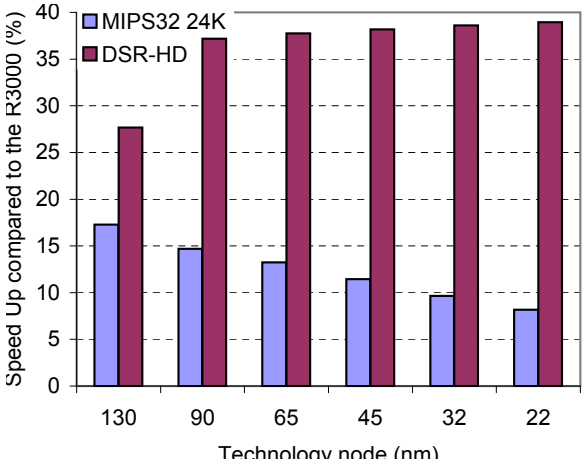

Figure I.1.108: bezier01fixed Benchmark Results

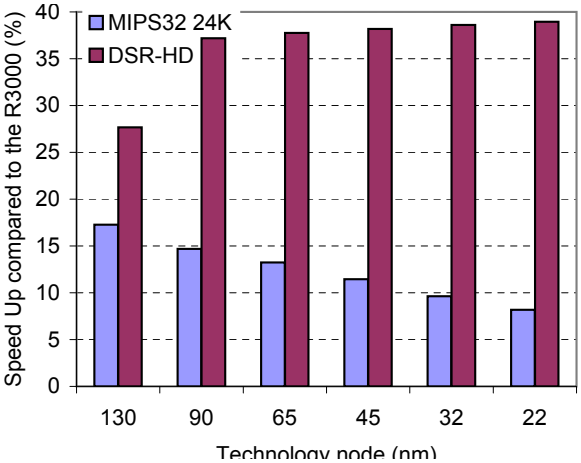

Figure I.1.109: bezier01float Benchmark Results

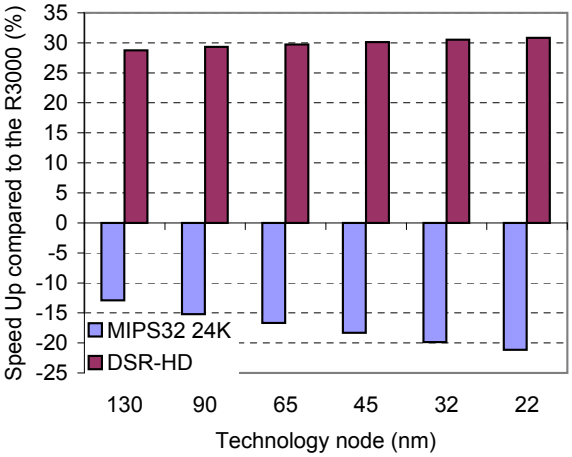

Figure I.1.110: dither01 Benchmark Results

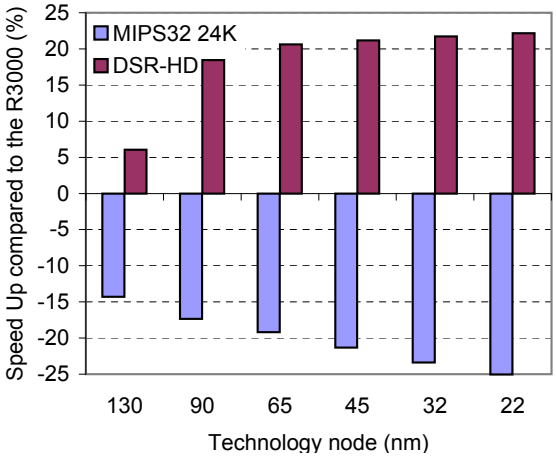

Figure I.1.111: rotate01 Benchmark Results

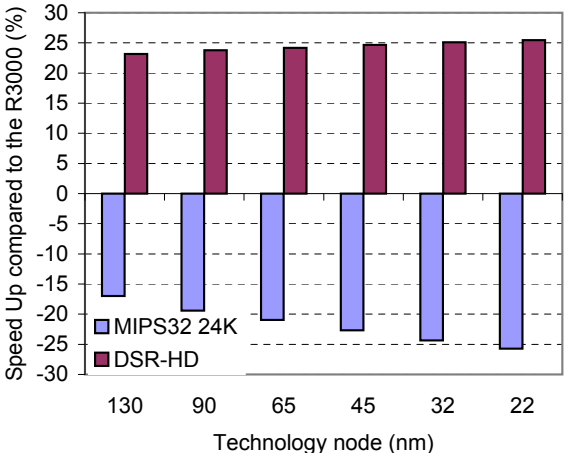

Figure I.1.112: text01 Benchmark Results

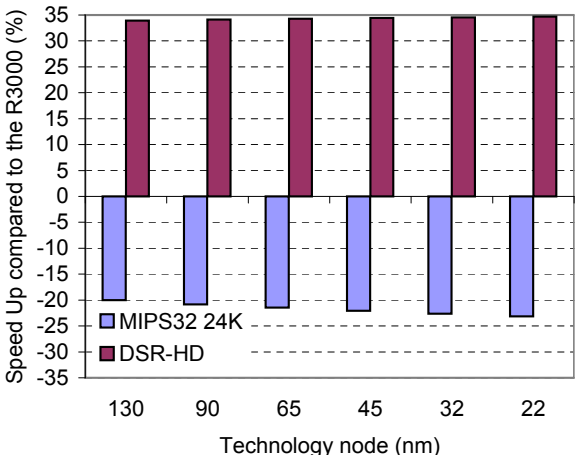

Figure I.1.113: djpeg Benchmark Results

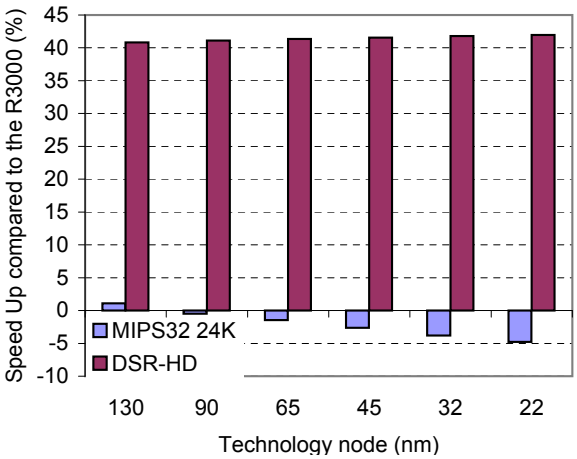

Figure I.1.114: rgbcmy01 Benchmark Results

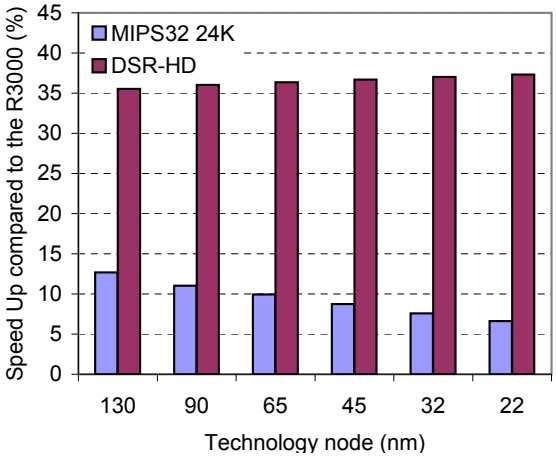

Figure I.1.115: rgbhpg01 Benchmark Results

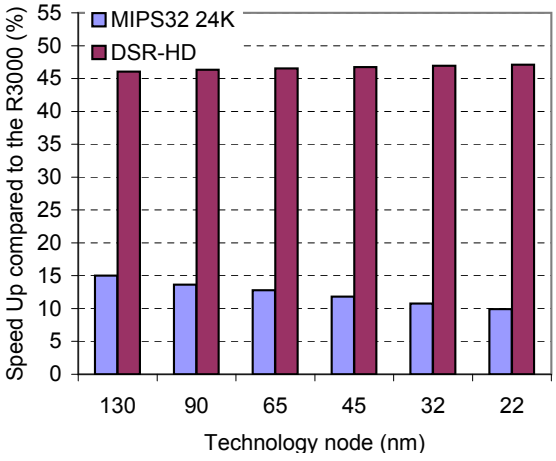

Figure I.1.116: rgbyiq01 Benchmark Results

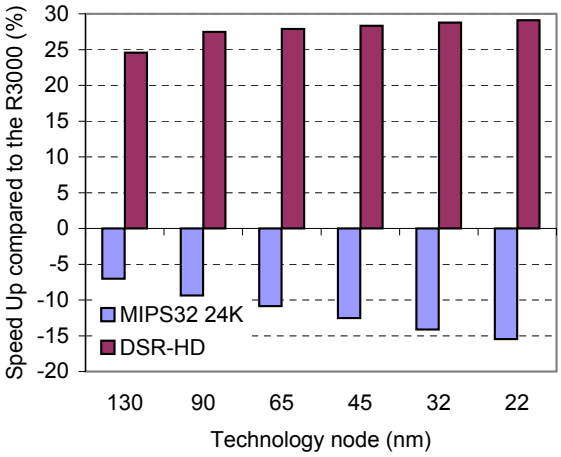

Figure I.1.117: ttsprk01 Benchmark Results

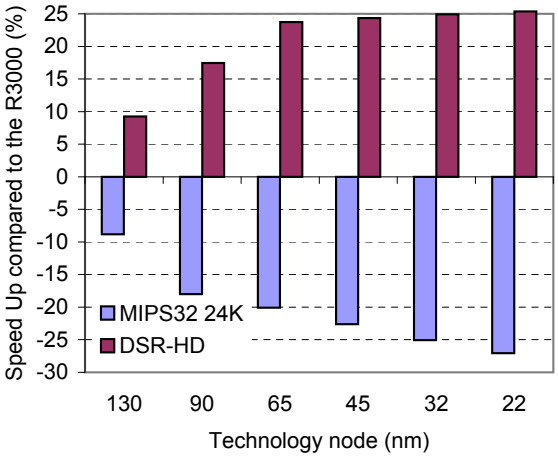

Figure I.1.118: tblook01 Benchmark Results

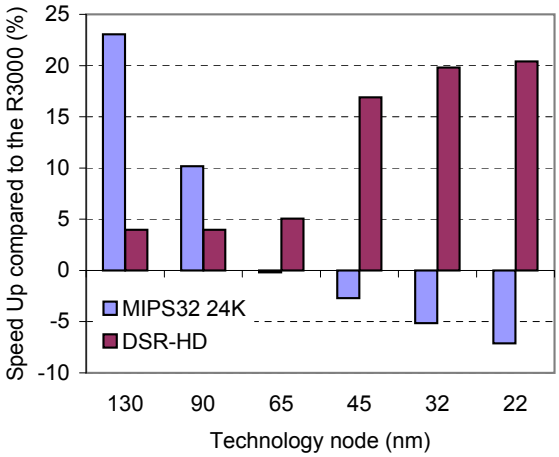

Figure I.1.119: rspeed01 Benchmark Results

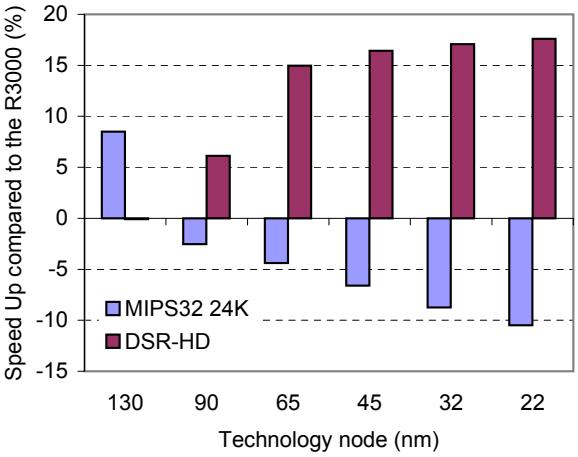

Figure I.1.120: puwmod01 Benchmark Results

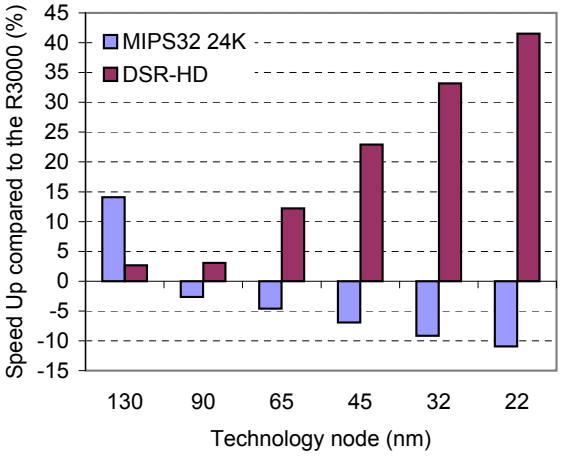

Figure I.1.121: pntrch01 Benchmark Results

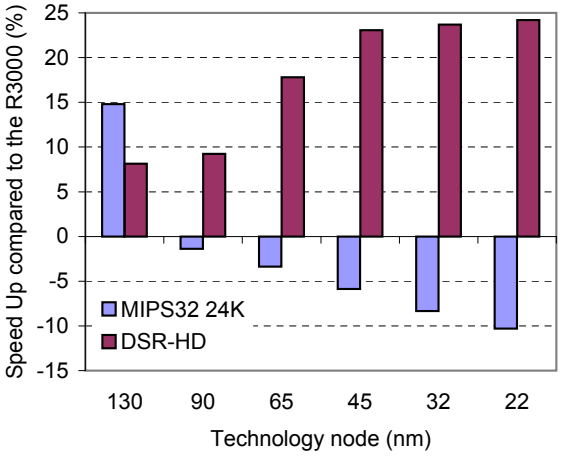

Figure I.1.122: canrdr01 Benchmark Results

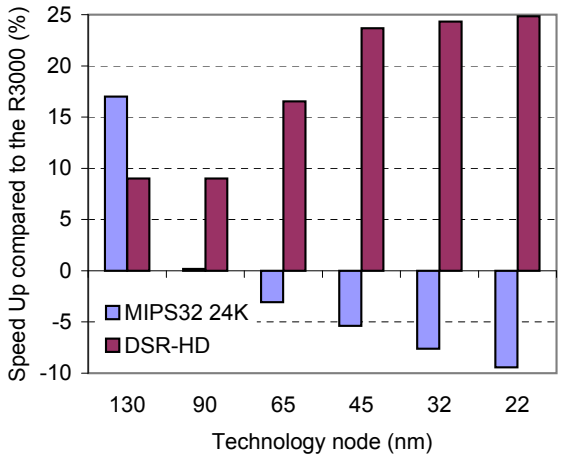

Figure I.1.123: cacheb01 Benchmark Results

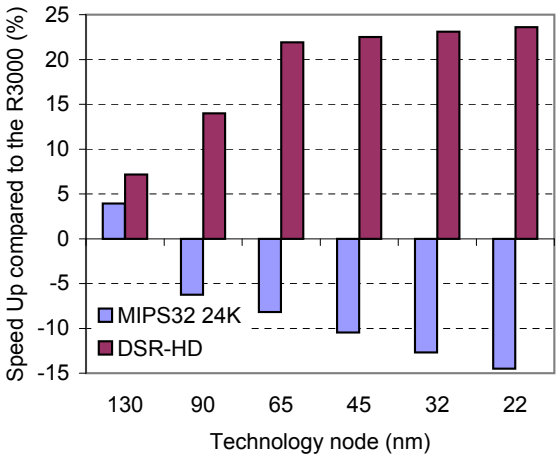

Figure I.1.124: bitmnp01 Benchmark Results

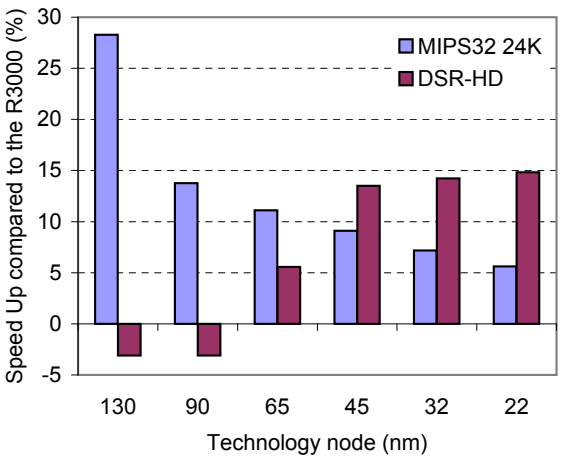

Figure I.1.125: aifirf01 Benchmark Results

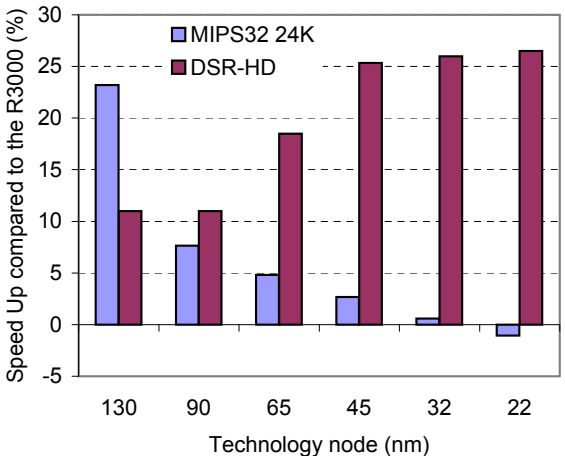

Figure I.1.126: a2time01 Benchmark Results

Appendix J

HD Processor Performance: Aggressive Wire Model

All the below graphs report the gain brought by the MIPS32 24K and DSR-HD processors over the MIPS R3000 on EEMBC benchmarks based on an optimistic wire load model. The core clock frequency was set to 350MHz for the MIPS R3000 processor and 625MHz for the eight-stage architectures.

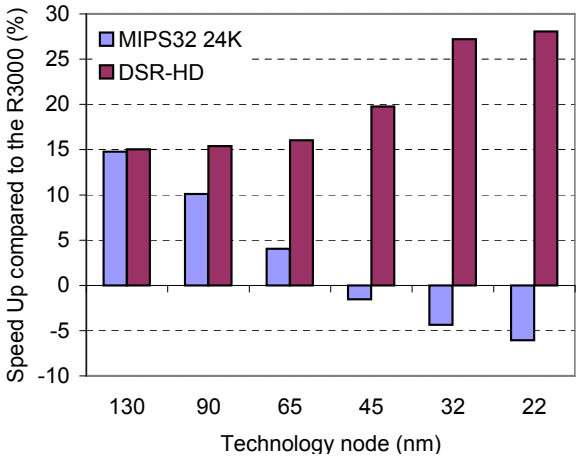

Figure J.1.127: Average Performance based on EEMBC Integer Benchmarks

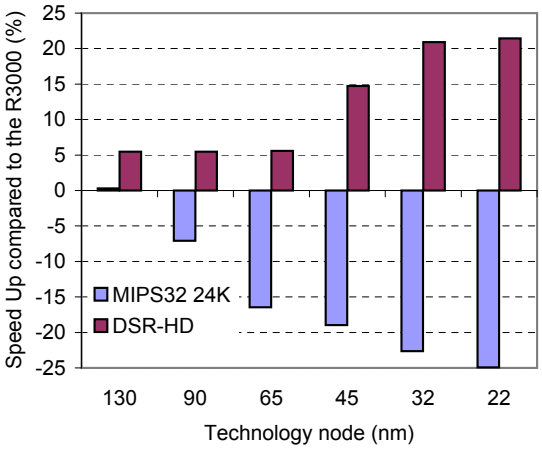

Figure J.1.128: ospf Benchmark Results

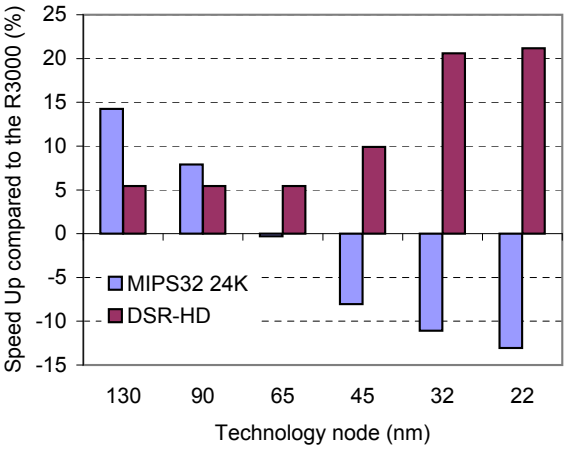

Figure J.1.129: pktflowb512k Benchmark Results

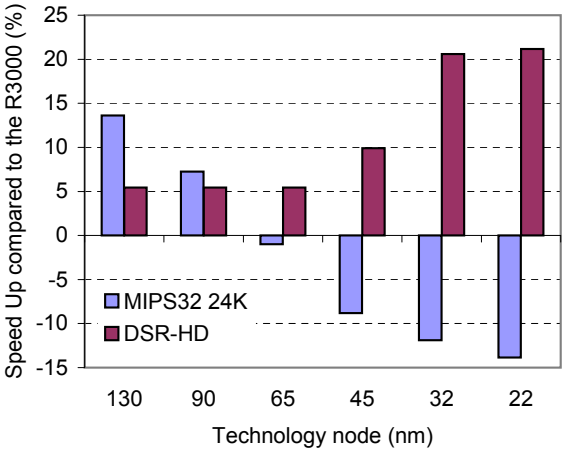

Figure J.1.130: pktflowb1m Benchmark Results

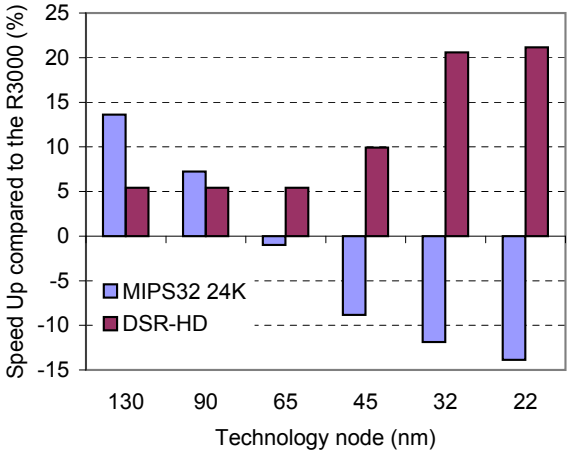

Figure J.1.131: pktflowb2m Benchmark Results

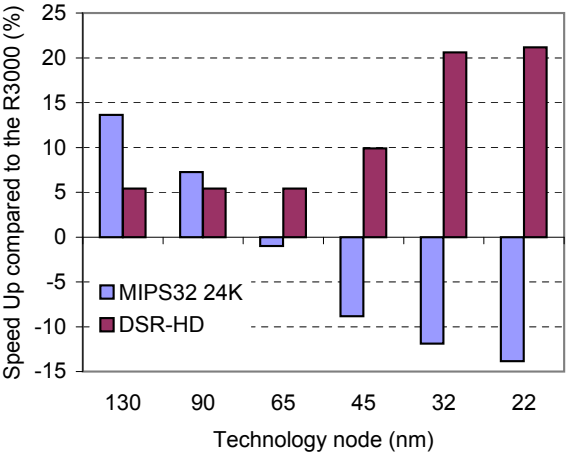

Figure J.1.132: pktflowb4m Benchmark Results

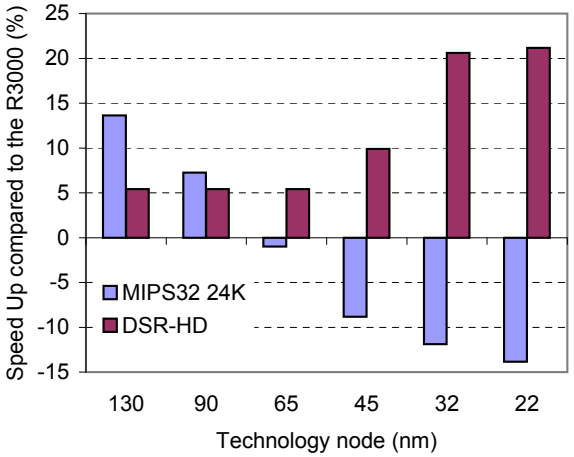

Figure J.1.133: routelookup Benchmark Results

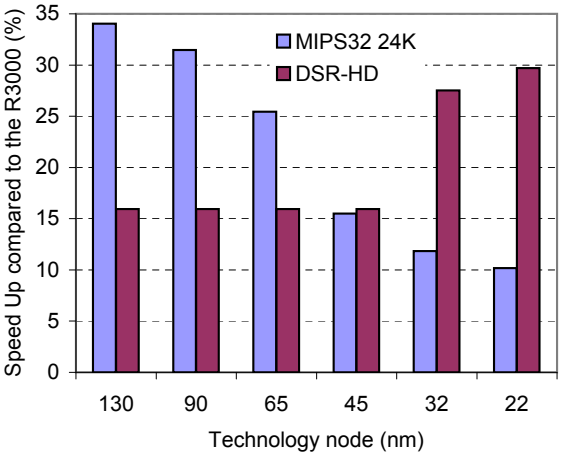

Figure J.1.134: autcor00data_1 Benchmark Results

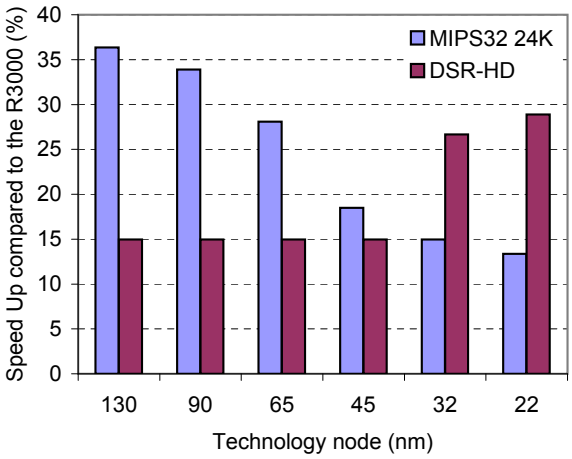

Figure J.1.135: autcor00data_2 Benchmark Results

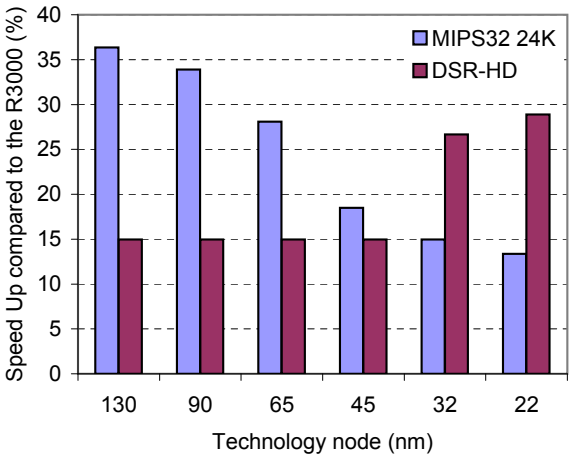

Figure J.1.136: autcor00data_3 Benchmark Results

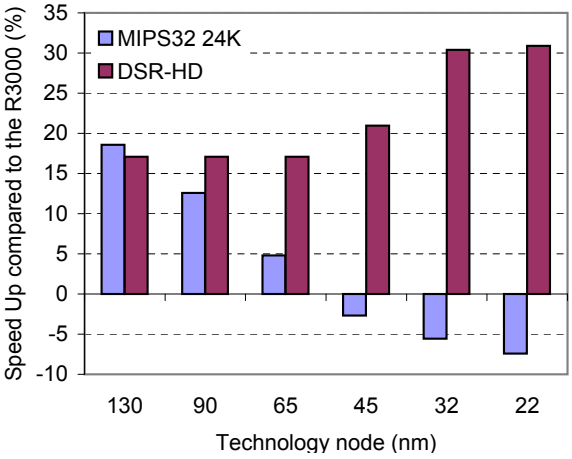

Figure J.1.137: conven00data_1 Benchmark Results

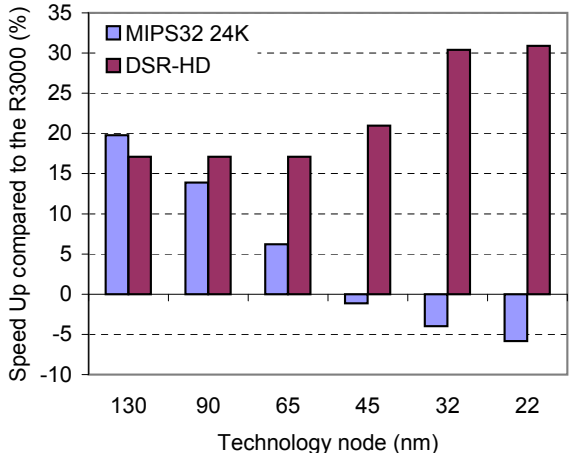

Figure J.1.138: conven00data_2 Benchmark Results

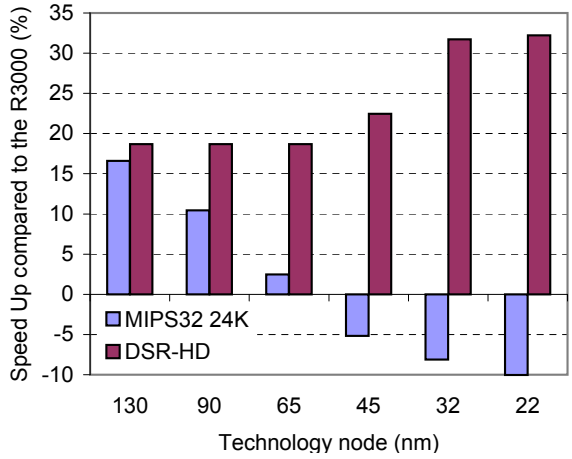

Figure J.1.139: conven00data_3 Benchmark Results

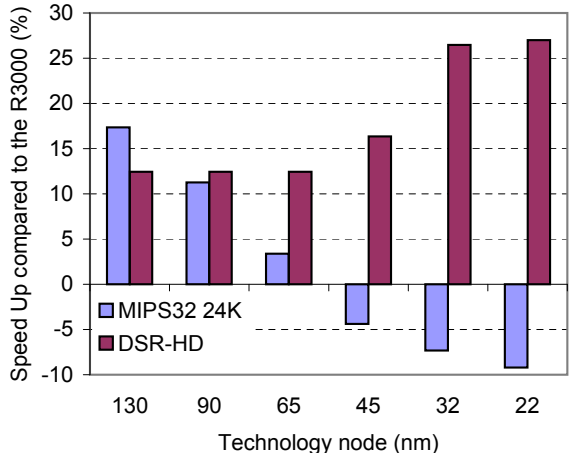

Figure J.1.140: fbital00data_2 Benchmark Results

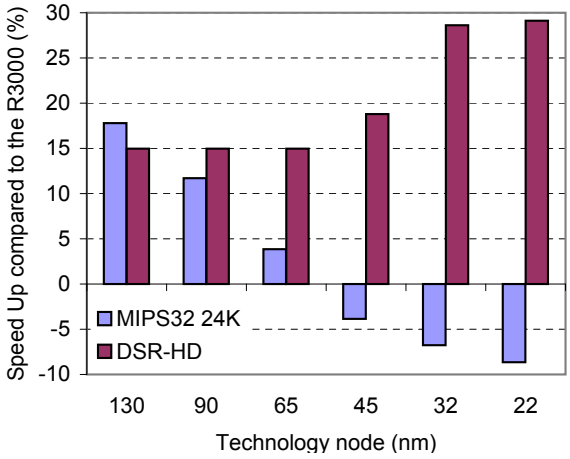

Figure J.1.141: fbital00data_3 Benchmark Results

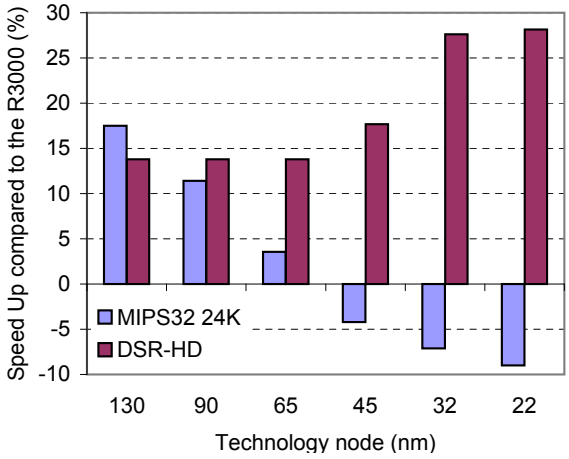

Figure J.1.142: fbital00data_6 Benchmark Results

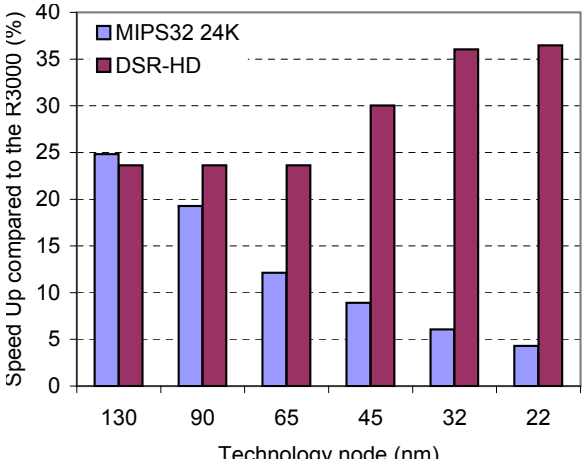

Figure J.1.143: fft00data_1 Benchmark Results

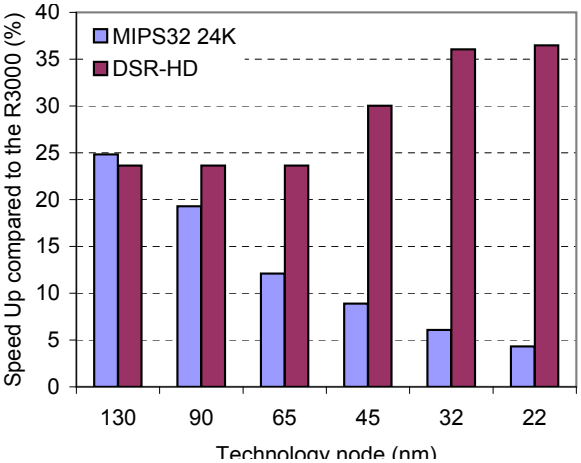

Figure J.1.144: fft00data_2 Benchmark Results

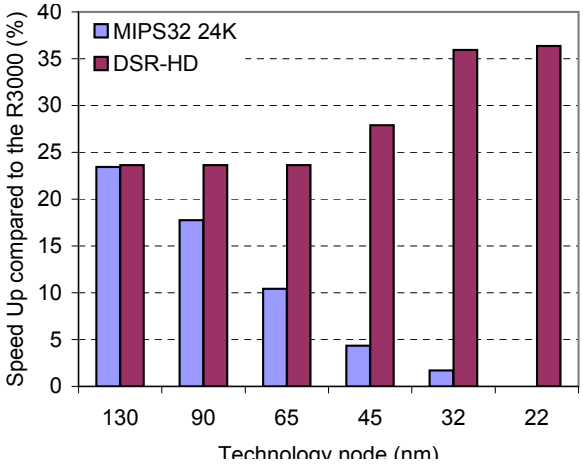

Figure J.1.145: fft00data_3 Benchmark Results

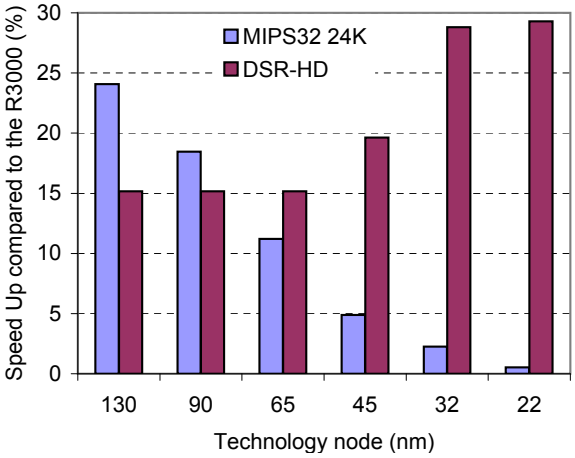

Figure J.1.146: viterb00data_1 Benchmark Results

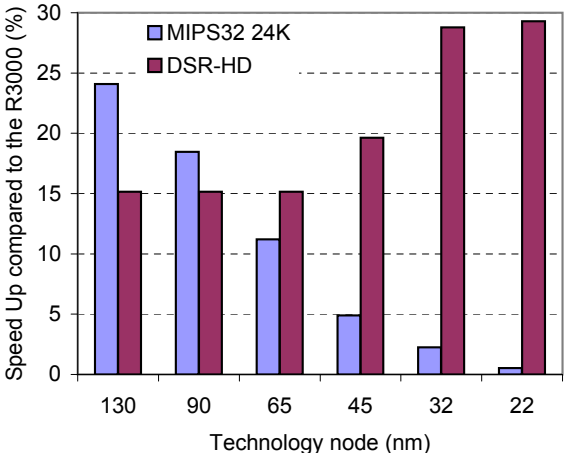

Figure J.1.147: viterb00data_2 Benchmark Results

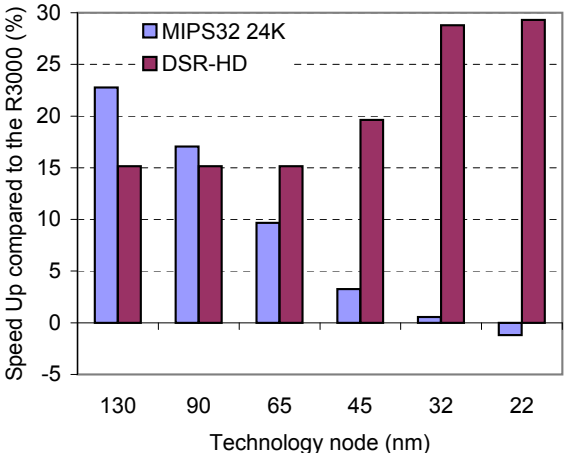

Figure J.1.148: viterb00data_3 Benchmark Results

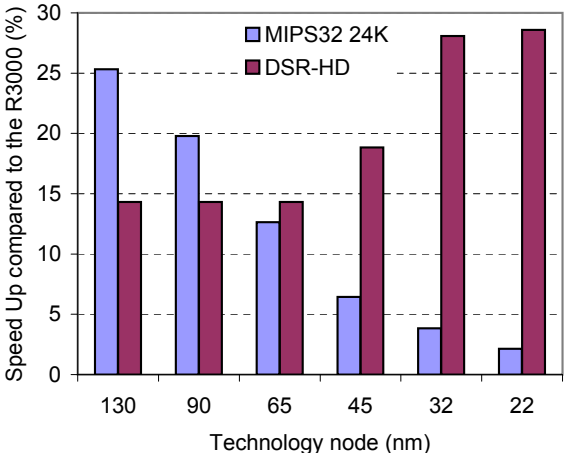

Figure J.1.149: viterb00data_4 Benchmark Results

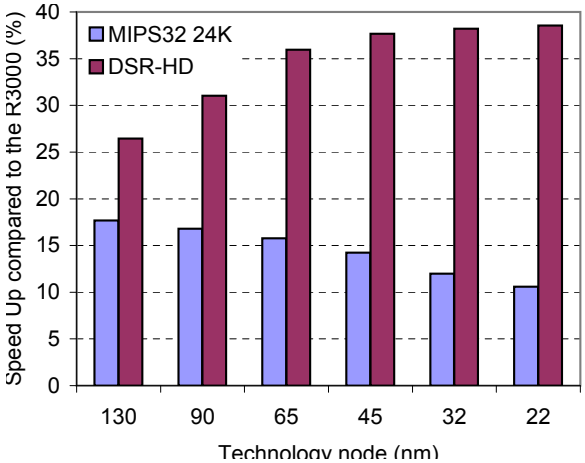

Figure J.1.150: bezier01fixed Benchmark Results

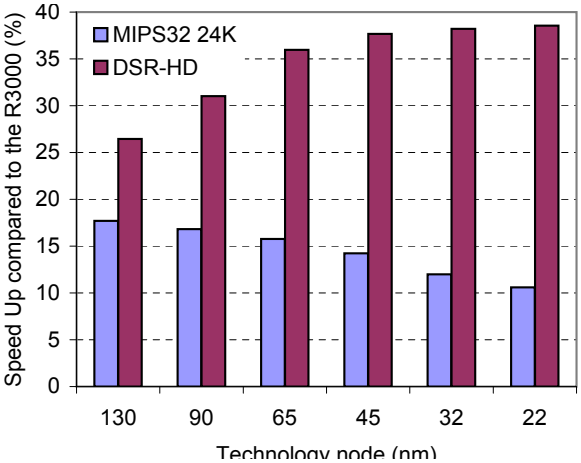

Figure J.1.151: bezier01float Benchmark Results

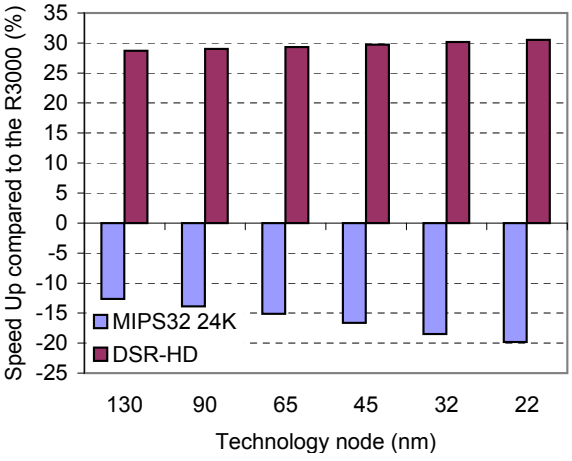

Figure J.1.152: dither01 Benchmark Results

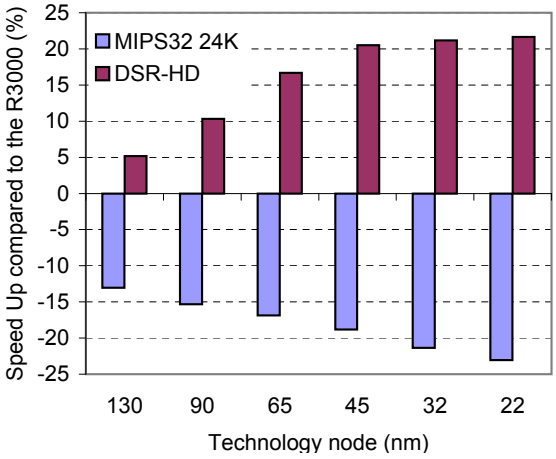

Figure J.1.153: rotate01 Benchmark Results

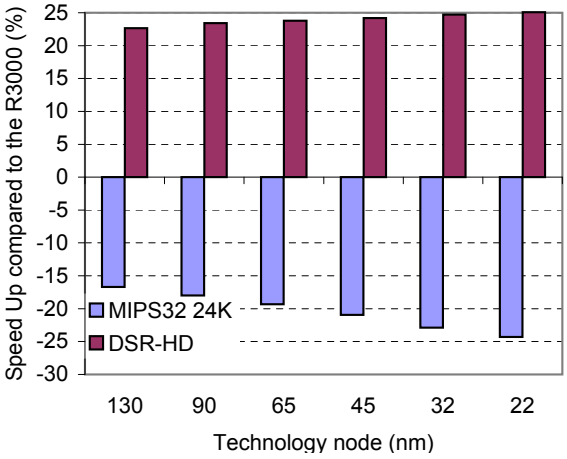

Figure J.1.154: text01 Benchmark Results

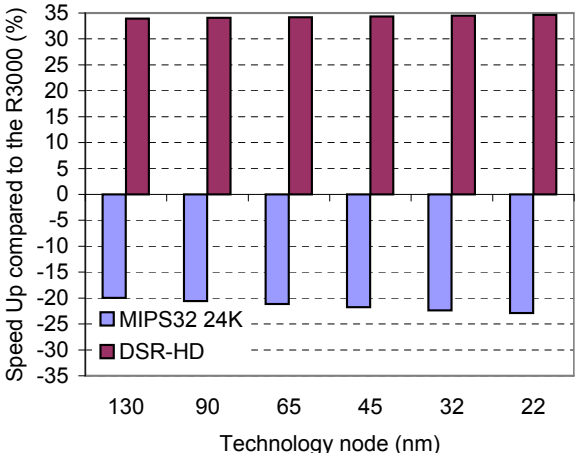

Figure J.1.155: djpeg Benchmark Results

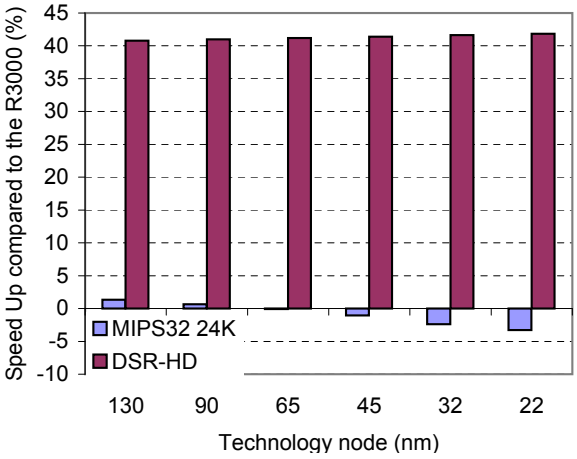

Figure J.1.156: rgbcmy01 Benchmark Results

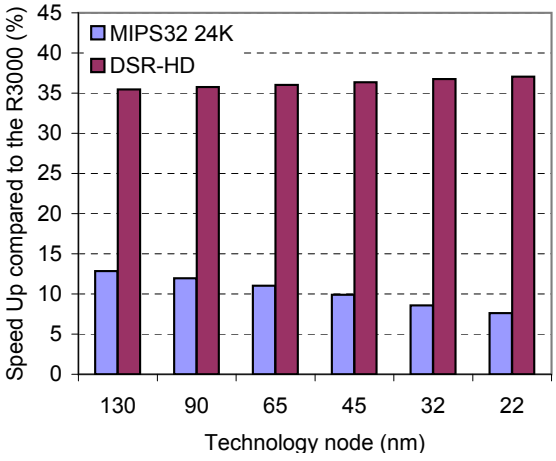

Figure J.1.157: rgbhpg01 Benchmark Results

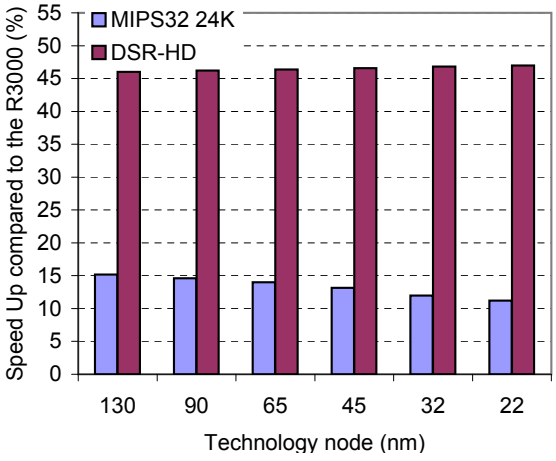

Figure J.1.158: rgbyiq01 Benchmark Results

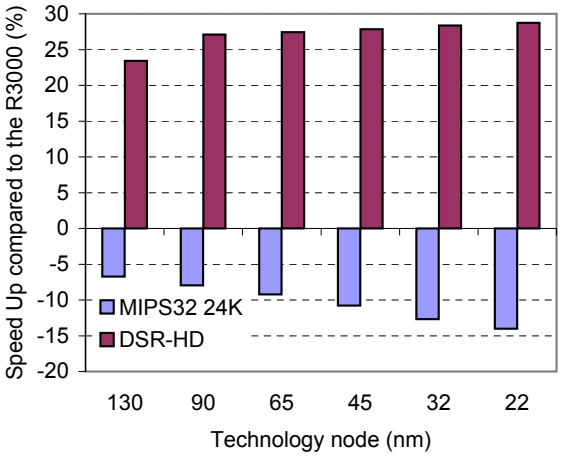

Figure J.1.159: ttsprk01 Benchmark Results

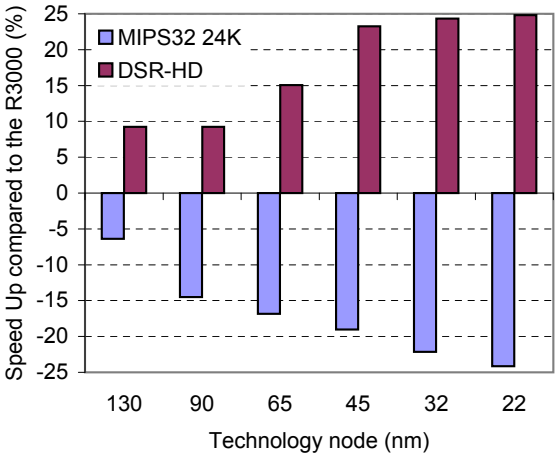

Figure J.1.160: tblook01 Benchmark Results

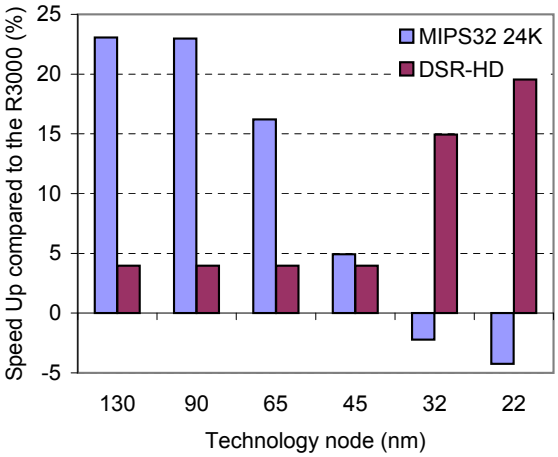

Figure J.1.161: rspeed01 Benchmark Results

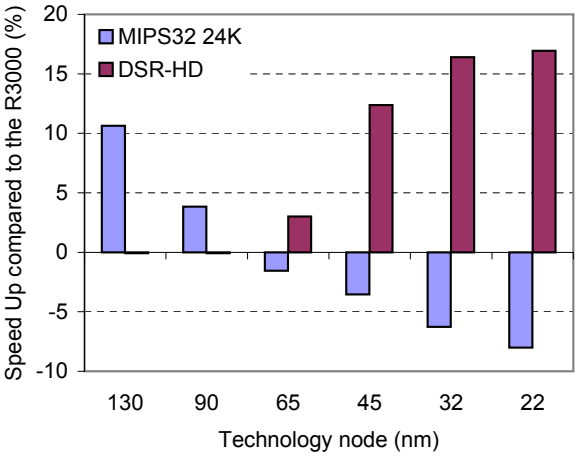

Figure J.1.162: puwmod01 Benchmark Results

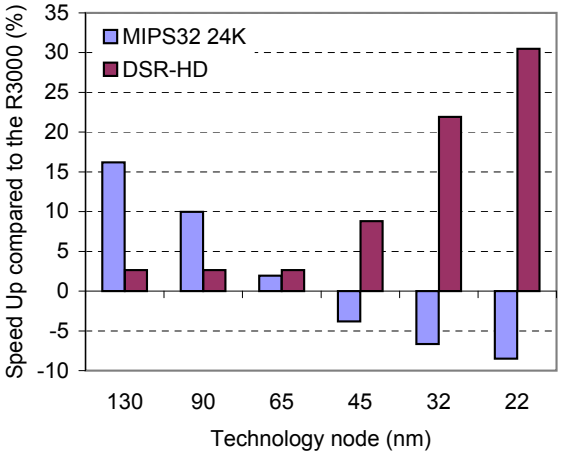

Figure J.1.163: pntrch01 Benchmark Results

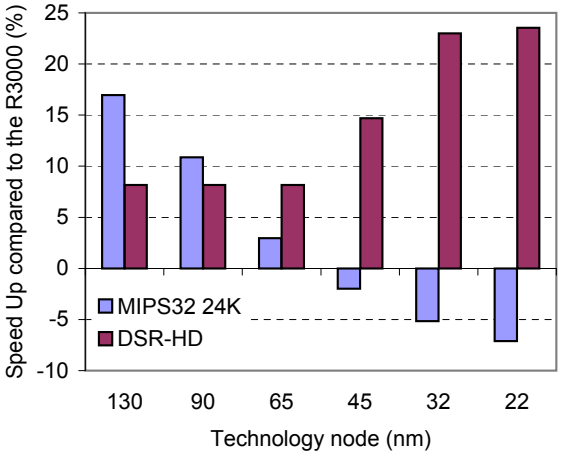

Figure J.1.164: canrdr01 Benchmark Results

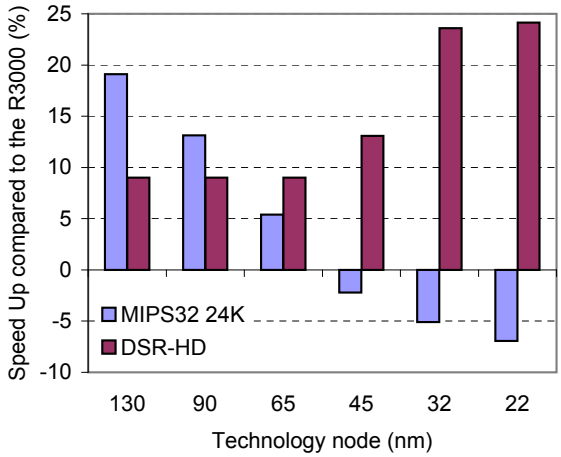

Figure J.1.165: cacheb01 Benchmark Results

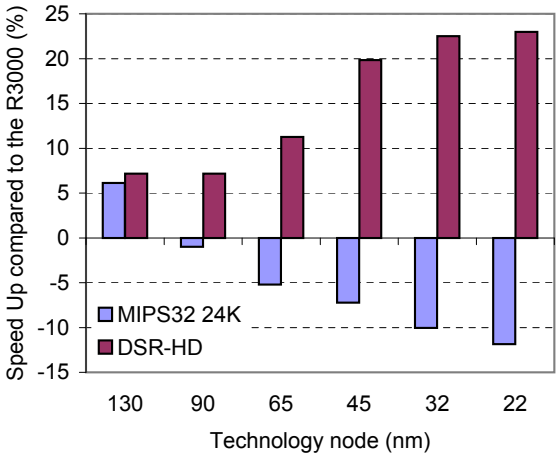

Figure J.1.166: bitmnp01 Benchmark Results

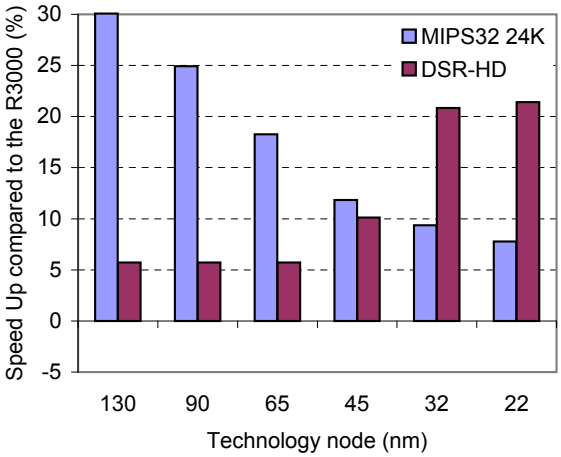

Figure J.1.167: aifirf01 Benchmark Results

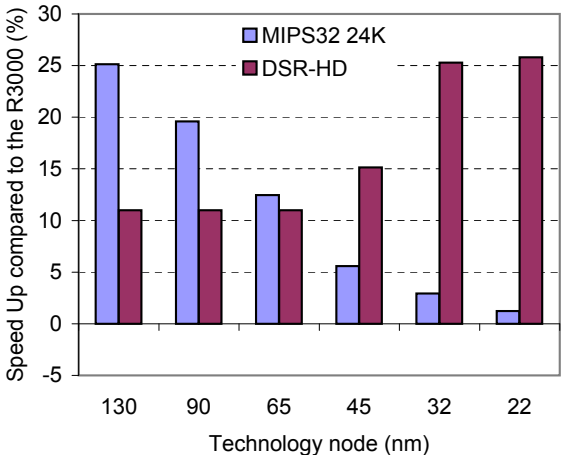

Figure J.1.168: a2time01 Benchmark Results

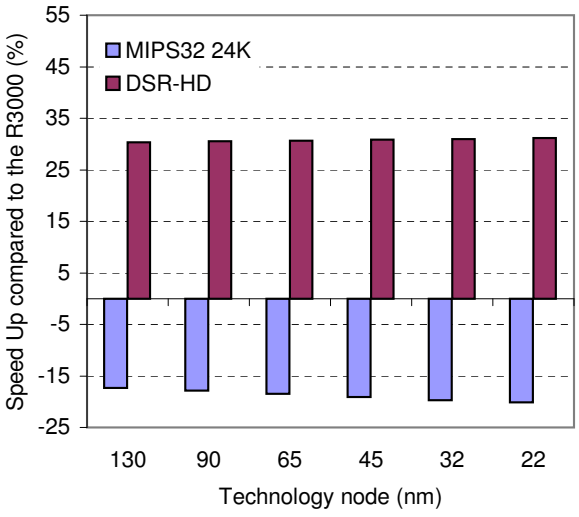

Figure J.1.169: cjpeg Benchmark Results

In this last part, the core clock frequency was set to 330MHz for the MIPS R3000 processor and 625MHz for the eight-stage architectures.

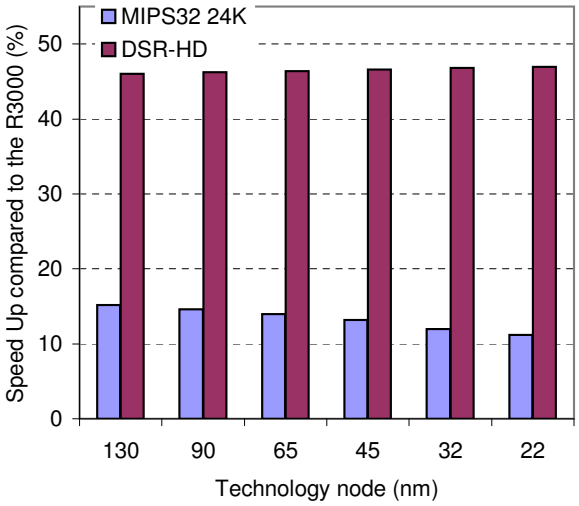

Figure J.1.170: Best DSR-HD Gains: rgbyiq01 Benchmark Results

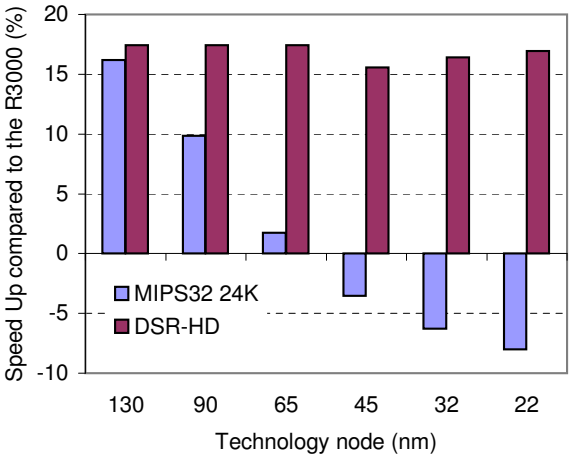

Figure J.1.171: Worst DSR-HD Gains: puwmod01 Benchmark Results

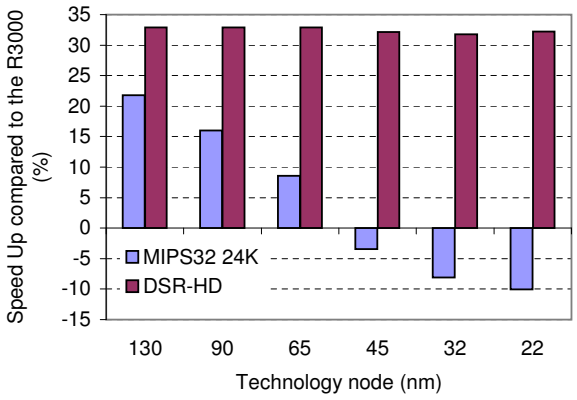

Figure J.1.172: Typical DSR-HD and MIPS32 24K Gains among all the EEMBC Applications: convendata00_3 Benchmark Results

Appendix K

HD Processor Efficiency: Aggressive Wire Model

All the below graphs report the gain brought by the MIPS32 24K and DSR-HD processors over the MIPS R3000 on EEMBC benchmarks based on an optimistic wire load model. The core clock frequency was set to 330MHz for the MIPS R3000 processor and 625MHz for the eight-stage architectures.

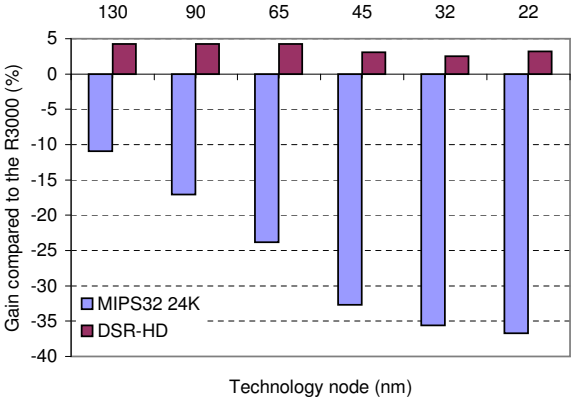

Figure K.1.173: Average Performance based on EEMBC Integer Benchmarks

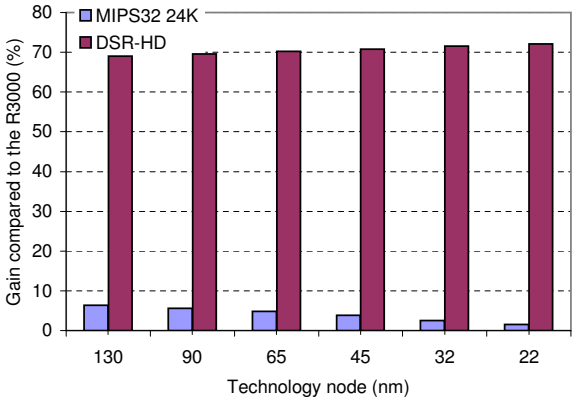

Figure K.1.174: rgbyiq01 Benchmark Results

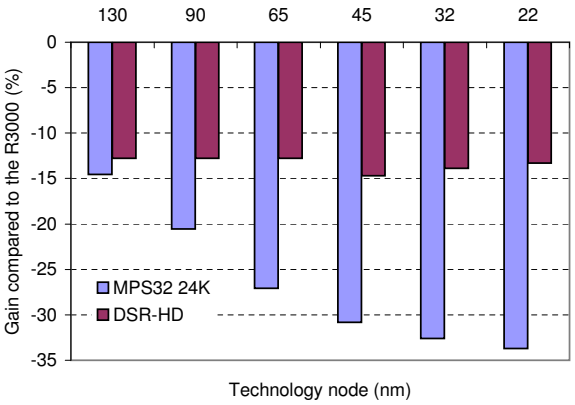

Figure K.1.175: puwmod01 Benchmark Results

Appendix L

Wire Delay Penalty

The gains of DSR-HD and the MIPS24k were established based on both aggressive and conservative wire load model and compared to the performance of the MIPS R3000 that uses an aggressive wire load model. Thus, the difference of the gains obtained, and reported in the figures below, highlight the effect of the wires on the performance. The core clock frequency was set to 330MHz for the MIPS R3000 processor and 625MHz for the eight-stage architectures.

It comes out that DSR-HD attenuates the most the increasing drawback of the wire delays.

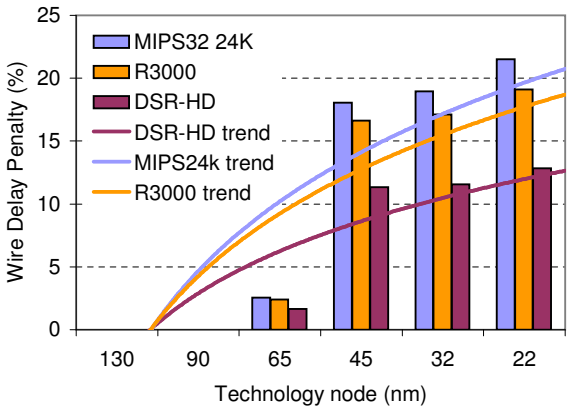

Figure L.1.176: Wire Delay Penalty on a Representative DSR-HD Performance Gain among all the EEMBC Applications: conven00data_3 Benchmark Results. The baseline is the execution time required by the MIPS R3000 processor with single port direct mapped primary memories to perform the conven00data_3 EEMBC benchmark and based on an aggressive wire load model.

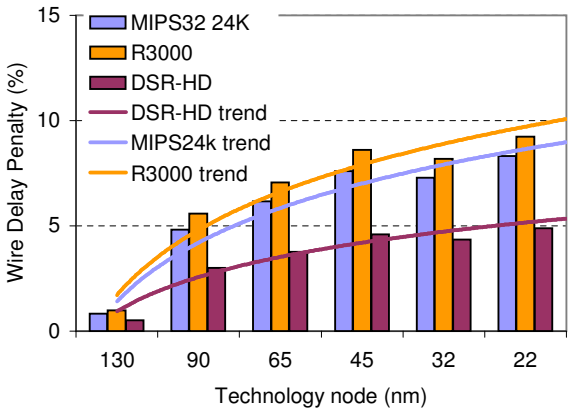

Figure L.1.177: rgbyiq01 Benchmark Results

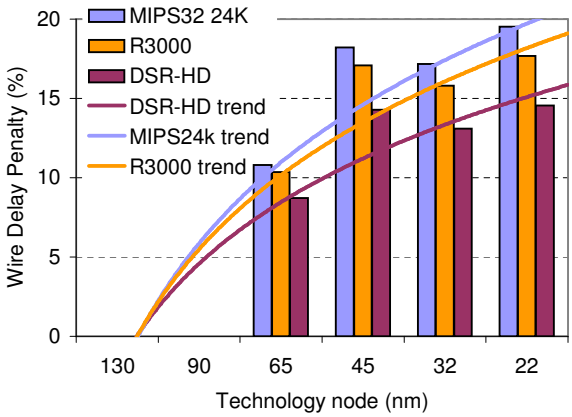

Figure L.1.178: puwmod01 Benchmark Results

Appendix M

Acronyms

AG	Address Generation
ALU	Arithmetic and Logic Unit
ARM	Advanced RISC Machine
ASIC	Application-Specific Integrated Circuit
ASIP	Application Specific Instruction set Processor
ASP	Application/Domain Specific Processor
ASSP	Application-Specific Standard Product
BP	Branch Predictor
BTIC	Branch Target Instruction Cache
CACTI	Cache Access and Cycle Time model
CAD	Computer Aided Design
CAM	Content Addressable Memory
CMOS	Complementary Metal Oxide Silicon
CP0	system control CoProcessor
CPI	Cycles Per Instruction
CPU	Central Processing Unit
CVS	Clustered Voltage Scaling
DC	Data Cache
DE	DEcode
DM	Direct Mapped
DP	Dual Port (memory)
DRAM	Dynamic RAM
DSP	Digital Signal Processor

DSR	Deep Submicron Architecture
DVS	Dynamic Voltage Scaling
EEMBC	Embedded Microprocessor Benchmarking Consortium
ER	Exception Resolution
FIFO	First In First Out
FO4	fanout-of-our inverter delay
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
FSF	Free Software Foundation
FSM	Finite State Machine
FSRAM	Flexible Sequential and Ramdon Access Memory
FMT	Fixed Map Translation
GAS	Gnu ASsembler
GCC	Gnu C Compiler
GDB	Gnu DeBugger
HD	High Density
HP	High Performance
IC	Instruction Count
IF	Instruction Fetch/Instruction fetch First
IS	Instruction fetch Second
ISS	Instruction Set Simulator/Instruction iSSue
ID	Instruction Decode
IP	Instruction Prefetch
ISA	Instruction Set Architecture
IT	Instruction fetch Third
ITRS	International Technology Roadmap for Semiconductors
LP	Low Power
LRU	Least recently Used
MA	Memory Access
MAC	Multiply and ACcumulate
MEM	MEMory
MF	Memory access First

MS	Memory access Second
MIPS	Microcomputer without Interlocked Pipeline stages
	OR
	Million Instructions Per Second
MMU	Memory Management Unit
NO	No Overlap
NOP	No OPeration
OS	Operating System
PDE	PreDEcode
PF	Program Fetch
PVT	Process Voltage Temperature
RA	Register Access
RAM	Random Access Memory
RAR	Read After Read
RAW	Read After Write
RISC	Reduced Instruction Set Computer
RF	Register File
RTL	Register Transfer Level
RTOS	Real Time Operating System
SAT	SATuration
SH	SHifter
SIT	Software Instrumentation Tool
SOC	System-On-Chip
SP	Single Port (memory)
SPEC	Standard Performance Evaluation Corporation
SRAM	Static RAM
SYNC	SYNChronization
TLB	Translation Lookaside Buffer
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLIW	Very Long Instruction Word
VMIPS	Virtual MIPS
VOS	Virtual Operating System
VSS	VHDL System Simulator

WAR Write After Read
WAW Write After Write
WB Write Back

Bibliography

[1] A. Agarwal, K. Roy, T.N. Vijaykumar, *Exploring High Bandwidth Pipelined Cache Architecture for Scaled Technology*, in Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, March 2003.

[2] V. Aggarwal, S. Aguirre, *Software Solutions for Single Instruction Issue, in Order Processors*, internal document of the University of Applied Science (Embedded Information Systems Institute), July 2004.

[3] T. Ahearn et al., *Virtual Memory System*, U.S. Patent No. 3781808, December 1973.

[4] D. Alpert, *Will Microprocessors Become Simpler?*, Microprocessor Report, http://www.mdronline.com/mpr/h/2003/1117/174603.html, November 2003.

[5] B. S. Amrutur, M. A. Horowitz, *Speed and power scaling of SRAMs*, IEEE Journal of Solid State Circuits, Vol.35, pp. 175-185, February 2000.

[6] ARM®, *ARM1136JF-S™ and ARM1136J-S™ Technical Reference Manual*, May 2004.

[7] J.-L. Baer, S. D. Pudar, *On the inclusion property for multi-level cache hierarchies*, in Proceedings of the 15th Annual Symposium on Computer Architecture, June 1988.

[8] R. Bai, S. Kulkarni, W. Kwong, A. Srivasta, D. Sylvester, D. Blaauw, *An Implementation of a 32-bit ARM Processor Using Dual Power Supplies and Dual Threshold Voltages*, IEEE International Symposium on VLSI, pp. 149-154, February 2003.

[9] P. Caputa, M.A. Anders, C. Svensson, R.K. Krishnamurthy, S. Borkar, *A Low-swing Single-ended L1 Cache Bus Technique for Sub-90nm Technologies*, in Proceedings of the European Solid-State Circuits Conference, pp. 475-477, Leuven, Belgium, September 2004.

[10] Y. Chen, K. Ranganathan, A. K. Puthenveetil, V. V. Pai, D. J. Lilja, K. Bazargan, *Enhancing the Memory Performance of Embedded Systems with the Flexible Sequential and Random Access Memory*, in Proceeding of the 9th Asia-Pacific Computer Systems Architecture Conference, September 2004.

[11] F. A. Cherigui, *High performance VLSI architectures implementing strong cryptographic primitives*, Ph.D. Thesis, Department of Electrical Engineering, Swiss Federal Institute of Technology, February 2002.

[12] Z. Chisti, T.N. Vijaykumar, *Wire delay is not a problem for SMT (in the near future)*, in Proceedings of the 31st Annual International Symposium on Computer Architecture, June 2004.

[13] J. Donovan, *MIPS Technologies and Virage Logic Announce Lowest Cost, Lowest Power, Highest Performance 333 MHz Embedded Processor*, Design And Reuse, January 2005.

[14] Faraday Technology Corp., *Faraday Introduces Microprocessor-Optimized SRAM*, Design And Reuse, April 2004.

[15] K. Fischer, *Embedded 32-Bit Microprocessors to Dominate Customer-Specific, Cell-Based Product Market*, In-Stat, http://instat.com, December 2003.

[16] S. Furber, *ARM System-on-chip Architecture*, Addison Wesley, Second Edition, 2004.

[17] T.R. Halfhill, *ARC 700 Secrets Revealed*, Microprocessor Report, http://www.mdronline.com/mpr/h/2004/0621/182501.html, June 2004.

[18] T.R. Halfhill, *Tensilica Tackles Bottlenecks*, Microprocessor Report, http://www.mdronline.com/mpr/h/2004/0531/182201.html, June 2004.

[19] M. Hamada, Y. Ootaguro, T. Kuroda, *Utilizing Surplus Timing for Power Reduction*, in Proceedings of the IEEE Custom Integrated Circuits Conference, pp. 89-92, 2001.

[20] J. L. Hennessy, D. A. Patterson, *Computer Architecture : a Quantitative Approach*, Morgan Kaufmann Publishers, Inc., Third edition, 2003.

[21] R. Ho, *On-Chip Wires: Scaling and Efficiency*, Ph.D. thesis, Department of Electrical Engineering, Stanford University, August 2003.

[22] R. Ho, K. Mai, M. Horowitz, *Managing Wire Scaling: A Circuit Perspective*, in Proceedings of the IEEE Interconnect Technology Conference, June 2003.

[23] R. Ho, K. W. Mai, M. A. Horowitz, *The future of wires*, in Proceedings of the IEEE journal in electrical engineering, Vol. 89, No. 4, pp. 490-504, April 2001.

[24] M. Horowitz, W. Dally, *How scaling will change processor architecture*, in Proceedings of the IEEE International Solid-State Circuits Conference, vol. 1, pp. 132-133, February 2004.

[25] M. S. Hrishikesh, N. P. Jouppi, K. I. Farkas, D. Burger, S. W. Keckler, P. Shivakumar, *The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays*, in Proceedings of the 29th International Symposium of Computer Architecture, pages 14–24, May 2002.

[26] F. Ishihara, F. SHeikh, B. Nikolić, *Level Conversion for Dual-Supply Systems*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, February 2004.

[27] G. Kane, J. Heinrich, *MIPS RISC Architecture*, Prentice-Hall, Inc., New Jersey, 1988.

[28] M. Koerner, C. Ming Fai, J. Ruthven, *Powerpc: An Inside View*, Prentice-Hall, Inc., New Jersey, 1996.

[29] M. Levy, *ARM Wrestles With MIPS - Comparing the ARM11 and MIPS 24K Microarchitectures*, Microprocessor Report, http://www.mdronline.com/mpr/h/2003/0804/173101.html, August 2003.

[30] M. Levy, *MIPS Pipeline Favors Synthesizability - Eight-Stage Pipeline Yields High Frequency*, Microprocessor Report, http://www.mdronline.com/mpr/h/2003/0630/172601.html, June 2003.

[31] G. W. McFarland, *CMOS Technology Scaling and Its Impact on Cache Delay*, PhD thesis, Stanford University, June 1997.

[32] S. S. McKee, *Reflections on the Memory Wall*, in Proceedings of the 1st conference on Computing frontiers, April 2004.

[33] MIPS Technologies Inc., *MIPS32® 24K™ Processor Core Family Software User's Manual*, September 2004.

[34] MIPS Technologies Inc., *Programming the MIPS32™ 24K™ Core Family*, September 2004.

[35] H. Metha, S. Aguirre, *A GCC-based Cross Compiler for Single Instruction Issue, In-order Processors*, internal document of the Swiss Federal Institute of Technology (Signal Processing Laboratory), July 2004.

[36] M. Nakajima, T. Yamamoto, S. Ozaki, T. Sezaki, T. Kanakogi, T. Furuzono, T. Sakamoto, T. Aruga, M. Sumita, M. Tsutsumi, A. Ueda, T. lchinomiya, *A 400MHz 32b embedded microprocessor core AM34-1 with 4.0GB/s cross-bar bus switch for SoC*, in Proceedings of the Solid-State Circuits Conference, February 2002.

[37] H. Nakamura, H. Okawara, S. Sakai, T. Boku, M. Kondo, in Proceedings of the International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, *SCIMA: A Novel Architecture for High Performance Computing*, November 1999.

[38] P. R. Panda, N. D. Dutt, A. Nicolau, *On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems*, ACM Transactions on Design Automation of Electronic Systems, vol. 5, July 2000.

[39] A. Papanikolaou, M. Miranda, F. Catthoor, H. Corporaal, H. De Man, D. De Roest, M. Stucchi, K. Maex, *Methodology for propagating technology trade-offs over memory modules to the application level*, In Program Acceleration through Application and Architecture driven Code Transformations PA3CT, pp. 71-73, September 2003.

[40] J.-K. Peir, W.W. Hsu, A. J. Smith, *Functional Implementation Techniques for CPU Cache Memories*, IEEE Transactions on Computers, vol. 48, pp. 100-110, February 1999.

[41] T. Sakurai, A. R. Newton, *Alpha-power Law Mosfet Model and its Applications to CMOS Inverter delay and Other Formulas*, in Proceedings of the Solid-state Circuits, Vol. 25, pp584-594, April 1990.

[42] R. Sam, *ZOOM: A Performance-Energy Cache Simulator*, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2002.

[43] A. Saulsbury, F. Pong, A. Nowatzyk, *Missing the Memory Wall: The Case for Processor/Memory Integration*, in Proceedings of the International Symposium on Computer Architecture, May 1996.

[44] P. Shivakumar, N. P. Jouppi, *Cacti 3.0: An integrated cache timing, power and area model*, Technical report, Compaq Computer Corporation, August 2001.

[45] T. Skotnicki, *CMOS Technologies for End of Roadmap - Nano CMOS*, internal document of STMicroelectronics, August 2004.

[46] A.J. Smith, *Cache Memories*, ACM Computing Surveys, September 1982.

[47] A. Stevens, *Level 2 Cache for High-performance ARM Core-based SoC Systems*, ARM's white paper, January 2004.

[48] D. Sweetman, *See MIPS run*, Morgan Kaufmann Publishers, San Francisco 1999.

[49] D. Sweetman, N. Stephens, *IDT R30xx Family Software Reference Manual*, IDT Manual, 1994.

[50] C. Tran & Co, *The MIPS32 24KE Core Family: High-Performance RISC Cores with DSP Enhancements*, Design and Reuse, http://www.us.design-reuse.com/news/news10387.html, May 2005.

[51] Wm. A. Wulf, Sally A. McKee, *Hitting the Memory Wall: Implications of the Obvious*, ACM Computer Architecture News, vol. 23, p.20-24, March 1995.

[52] B. Zhai, D. Blaauw, D. Sylvester, K. Flautner, *Theoretical and Practical Limits of Dynamic Voltage Scaling*, in Proceedings of the 41st annual ACM IEEE Design Automation Conference, June 2004.

[53] ARC 600-700 families, http://www.arc.com

[54] ARM, http://www.arm.com

[55] The Embedded Microprocessor Benchmark Consortium, http://www.eembc.com

[56] B. Gaeke, *The VMIPS Project*, Version 1.1.3, http://www.dgate.org/vmips

[57] GCC home page - cross compiler tool chain, http://gcc.gnu.org

[58] IBM, http://www.ibm.com

[59] In-Stat - Microprocessor Report, http://www.in-stat.com

[60] International Technology Roadmap of Semiconductors 2003, http://public.itrs.net

[61] Microprocessor Directory 2005, http://www.edn.com

[62] MIPS Technologies Inc., http://www.mips.com

[63] Synopsys Inc., http://www.synopsys.com

[64] Taiwan Semiconductor Manufacturing Company, http://www.tsmc.com

[65] Tensilica Inc., http://www.tensilica.com

[66] Virage Logic, http://www.viragelogic.com

AGUIRRE Sylvain
Route du village
1041 Naz - Switzerland
+41-78-739-95-39

32 years old
Maried
French
sylvain.aguirre@a3.epfl.ch

Microelectronics Engineer

OBJECTIVE

Manager of an integrated circuit design team.

❑ EXPERIENCE

2003-2005 **Project leader** at the Engineering High School (EIVD) of Yverdon, Switzerland, and for a telecom company, Transwitch (Connecticut, USA) in Lausanne:
Creation of **a high performance embedded processor** for network applications.

2002 **Software engineer** at Transwitch (Connecticut, USA):

- **Elaboration of a telecom simulator** (UML, SystemC, C_{++}) in order to improve the VLSI system design flow.
- **Hardware design** (VHDL, VSS, CVS) and **verification** (coverage) of a **DDR-SDRAM memory controller**.

2001 **Hardware engineer** in the Integrated System Laboratory at the Swiss Federal Institute of Technology (EPFL) in Lausanne, Switzerland :
Conception (Synopsys) and verification (Xilinx) of **low power integrated circuits** that implement **the new cryptographic standard algorithm**, **Advanced Encrypted Standard** (AES).

2000 3 months internship at **Thomson-CSF** and **Cadence** in Paris, France:
Development of a VHDL simulator (Hal) expected to **improve FPGA design flow**.

❑ COMPUTER SCIENCE

Methodologies	: UML, CVS, Grafcet, DNAnet.
Operating Systems	: Unix, Linux, Windows, MS-DOS, QNX.
Computer science languages	: VHDL, Verilog, SystemC, C_{++}, C, $flex_{++}$, $bison_{++}$, assembler.
Software	: Synopsys, Mentor Graphics, Xilinx, Altera, Cadence, Spice, Matlab-Simulink.
Protocols and Algorithms :	AES, DES, RSA, TCP/IP, IPsec, MIPS ISA.

❑ LANGUAGES

French: native language **English**: fluent **Spanish**: good

❑ EDUCATION

2002-2006 **Thesis** at the EPFL: "**Deep-submicron Embedded Processor Architectures for High-performance, Low-cost and Low-power**". Expected graduation: April 2007.

2004 **Management of Innovation and New Technology** graduate diploma at the EPFL.

1997-2001 **Engineer diploma in microelectronics and robotics** (5 year degree) at Montpellier University, France.

❑ ACTIVITIES

Sports & leisure Tennis instructor, treasurer of a tennis club, organization of tennis tournaments. Member of a tennis team: national championship, coach of a soccer club.